2015 PICAXE MICROCHESS

PICAXE Microchess

Ian Mitchell

PICAXE MICROCHESS

This document describes the process of porting version one of Microchess originally developed on and for the KIM-1 6502 development system to the PICAXE 28X2 and 20X2 microcontrollers. The resulting program is logically equivalent to the original software

Contents

Abstract

This document describes the process of porting version one of Microchess originally developed on and for the KIM-1[7] 6502 development system to the PICAXE[6] 28X2 and 20X2 microcontrollers. The resulting program is logically equivalent to the original software with some user-friendly enhancements enabled through some of the features of the PICAXE devices.

Introduction

Microchess was developed in 1976 originally for the KIM-1 by Peter Jennings[1]. It was written in 6502 assembly language and needed only 924 bytes for the program. When playing, moves were entered using the hexadecimal keyboard and displayed on the six digit seven segment display. Microchess was subsequently ported to several microprocessor systems including the Apple I, Apple II, TRS-80, Commodore PET, Atari, etc.

In 2005 the program was ported to C[2] and also to the Arduino platform in 2014[3]. These authors took a different approach to porting the program. In the C version each 6502 mnemonic was defined as a sequence of C instructions and the compiled version was therefore the logical equivalent of the original Microchess program. As such it is able to run on any system supported by a C compiler. The Arduino version, also written in C, implements a 6502 emulator and executes the machine code of the original program.

The C program presented here takes Forster's idea but instead of generating a compiled logical version of the original program it generates a PICAXE BASIC file that is then uploaded into a 28X2 for execution. There were several issues that needed to be addressed in order for the Microchess program to be ported and to successfully preserve the original program logic as faithfully as possible in order to run on the 28X2 (and the 20X2).

Some of the issues that needed to be resolved included:

- no carry flag - this affects the 6502 add, subtract, compare and logical shift and rotate instructions
- no stack - this affects the 6502 push and pop instructions
- limited subroutine stack size - the 6502 has an 8 bit stack theoretically allowing for subroutines to be nested 128 deep (two bytes per return address). The PICAXE has a subroutine stack size of 8 whereas the Microchess program needs a stack at least 24 deep.

Some of the features of the 28X2 that were used to overcome the difficulties of the Microchess implementation are listed below:

- 64MHz clock
- 4096 bytes program memory (although there are 4 program slots of 4k each)
- 1024 bytes general purpose RAM
- 56 bytes directly accessible registers that can be overlaid on 28 16 bit words
- memory pointer register with auto increment and decrement
- 16 bit arithmetic.

A 20X2 version was also implemented but required modification of the zero page access addresses.

Hardware Implementation

The 28X2 version was developed and tested using an AXE401 Shield Base[4]. For operation at 64MHz a 16MHz crystal was installed at X1. Capacitors were found not to be required but they should probably be installed. The 20X2 version was developed and tested using an AXE118[5] connected to a 4.5 volt battery.

Software Implementation

For simplicity the carry flag was implemented in a separate register. To determine the carry from the arithmetic instructions the word size feature of the PICAXE was used. The high byte can be used to determine the state of the carry flag by arranging the registers of arithmetic instructions to generate a 16 bit result. For example the ADC (immediate) instruction was implemented as follows:

```
reg_fc = reg_a+0xhh+reg_cy
reg_a = reg_f
```

where 0xhh is the immediate value in hexadecimal and reg_fc is the 16 bit combination of reg_f and reg_cy.

The SBC instructions were implemented using the same one's complement trick from the 6502 – by taking the logical complement of the operand and adding:

```
get 0xhh,reg_f
reg_f = not reg_f
reg_fc = reg_a+reg_f+reg_cy
reg_a = reg_f
```

where 0xhh is the zero page location in hexadecimal of the operand stored in the 28X2 scratch RAM.

To overcome the limited return stack depth in the PICAXE series each return location is assigned a label and an associated reference number is stored using the scratch memory pointer. An advantage of this method is that the return address uses only a single byte of storage. For example the 6502 JSR SNGMV instruction is implemented as follows:

```
      @ptrdec = 7 : goto _SNGMV_
_07:  if reg_f!=0 then goto _KING_
```

The return instruction is implemented by retrieving the return address reference number using the scratch memory pointer and via the *branch* instruction execution is returned to the correct program location. The 6502 RTS instruction is implemented as follows:

```
__return__:
      inc ptr
      branch @ptr,(_00,_01,_02,_03,_04,_05,_06,_07,…)
```

On the 20X2 several program changes were required in order to accommodate the relatively limited memory available. All of the zero page references starting at 0x50 were relocated to 0x00. As a side effect this resulted in a small efficiency over the 28X2 implementation since it obviates the need for the offset required to access the piece location table. An instruction was inserted in the MOV

subroutine to force the memory access through register X to an unused location in 20X2 scratch memory. The original Microchess program uses two stacks at 0x01ff and 0x01c8. This was changed to 0x7f and 0x60 to accommodate the 128 bytes available in scratch RAM on the 20X2.

The 28X2 compiled to 3976 bytes whilst the 20X2 program compiled to 3836 bytes.

Implementation Enhancements

Some minor enhancements were implemented in the PICAXE version in order to make the game a little easier to control. To allow for changing the level of play from the command prompt, the two LDA and CMP immediate instructions that set the depth of search were changed to load from zero page and labelled LEVEL1 and LEVEL2. In the original version when the board is cleared the REV flag is not taken into consideration. If the board has been reversed (using the exchange command) and then cleared (using the clear command) the queen and king are transposed. By clearing the REV flag when the board is cleared this issue is avoided. The original version would also display the complete board as the position is entered digit by digit. This was changed so that the board is only displayed after entering a complete move.

Other Features Implemented

Since the PICAXE has 256 bytes of EEPROM for non-volatile storage this was taken advantage of by allowing the board to be saved, loaded and restored (after an unexpected power outage for example).

Implementation Optimisations

During coding of the 6502 instructions it became apparent that some optimisations could be implemented if they were applied with care. In the original program the stack is reset at the start of the main loop. This is easily achieved as a side effect of the way the stack is implemented in the PICAXE scratch RAM and using the *ptr* variable as the stack pointer. Some subroutines can be called using the PICAXE native call mechanism if they are completely self contained and do not make any recursive calls. Many 6502 instructions affect the status register which needs to be taken into account in any emulation mechanism. However there are usually many cases where the statuses are not required or are overwritten immediately (by the next instruction for example). In these cases it is not strictly necessary to update the status flags in every instruction. For example the LDA #ff immediate instruction is usually implemented as follows: `reg_a = 0xff : reg_f = reg_a`. However if the zero and negative flags are never tested after this instruction then is can be optimised as follows: `reg_a = 0xff.`

All optimised 6502 instructions are identified with a trailing underscore, e.g. LDAi_, JSR_, RTS_ etc.

Compiling the C Source

The C source can be compiled and executed on Windows or Linux. On Windows, Microsoft Visual C++ Express was used to create a Win32 console application with no initial source code. The Microchess source file is then added to the project which can then be compiled. The resulting executable can then be run from the windows command prompt.

To compile and run on Linux use the following commands:

```
g++ -o 28X2Microchesss 28X2Microchesss.cpp
./28X2Microchess
```

Summary of Commands

The user interface of the PICAXE version of Microchess uses the serial programming pins to input moves and output the board. Both the 28X2 and the 20X2 are configured to operate at 64MHz so the serial port operates at 76800 baud. Below is a summary of the commands accepted (they are not case sensitive):

Key	Command	Description
0-7	Rank or file number	Rank or file of piece to move
E	Exchange or reverse	Swap pieces with computer
CR	Carriage return	Move piece selected
C	Clear the board	Set up the board and clear the reverse flag
X	Set level 1	Set computer play to level 1 (super blitz)
Y	Set level 2	Set computer play to level 2 (blitz)
Z	Set level 3	Set computer play to level 3 (normal)
S	Save position	Save the board, reverse flag and play level in EEPROM
L	Load position	Load the previously saved position from EEPROM
R	Restore position	Restore the position after a reset or power loss
U	Undo move	Restore the board to the previous state
P	Play chess	Instruct the computer to calculate and play its move

Conclusion

Microchess for the KIM-1 was successfully ported to the PICAXE platform. The technique employed could be applied to other software and on other platforms.

Acknowledgements

Thanks to Peter Jennings for creating a remarkable piece of software and for permission to publish this version of MICROCHESS for the PICAXE microcontroller.

References

1. http://www.benlo.com/microchess
2. http://www.benlo.com/microchess/ForsterMicrochessC.zip
3. http://obsolescence.wix.com/obsolescence#!kim-uno-microchess/c1uxh
4. http://www.PICAXE.com/Hardware/Project-Boards/PICAXE-28X2-Shield-Base-Kit/
5. http://www.PICAXE.com/Hardware/Project-Boards/PICAXE-20-Project-Board/
6. http://www.PICAXE.com/What-Is-PICAXE
7. http://en.wikipedia.org/wiki/KIM-1
8. Garold R. Stone, An Upgrade for KIM MICROCHESS 1.0, Compute II, Issue 1, April/May 1980, Page 19 (http://www.atarimagazines.com/computeii/issue1/page19.php)
9. http://en.wikipedia.org/wiki/Microchess
10. http://en.wikipedia.org/wiki/1K_ZX_Chess

11. Mitchell, IG, SC3 - Single Chip Chess Computer, Electronics Australia, Consolidated Press, Sydney, Australia, 58, 6, June (1995) [Magazine Article] (http://ecite.utas.edu.au/53468)

Appendix A – Example Game

Example game on level 2 (blitz) with Microchess playing black:

```
   00 01 02 03 04 05 06 07
  ------------------------
00|WR|WN|WB|WK|WQ|WB|WN|WR|00
  ------------------------
10|WP|WP|WP|WP|WP|WP|WP|WP|10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|  |**|  |**|  |**|  |30
  ------------------------
40|  |**|  |**|  |**|  |**|40
  ------------------------
50|**|  |**|  |**|  |**|  |50
  ------------------------
60|BP|BP|BP|BP|BP|BP|BP|BP|60
  ------------------------
70|BR|BN|BB|BK|BQ|BB|BN|BR|70
  ------------------------
   00 01 02 03 04 05 06 07
CC CC CC
?
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|BB|BQ|BK|BB|BN|BR|00
  ------------------------
10|BP|BP|BP|BP|BP|BP|BP|BP|10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|  |**|  |**|  |**|  |30
  ------------------------
40|  |**|  |**|  |**|  |**|40
  ------------------------
50|**|  |**|  |**|  |**|  |50
  ------------------------
60|WP|WP|WP|WP|WP|WP|WP|WP|60
  ------------------------
70|WR|WN|WB|WQ|WK|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
EE EE EE
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|BB|BK|BQ|BB|BN|BR|00
  ------------------------
10|BP|BP|BP|BP|BP|BP|BP|BP|10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|  |**|  |**|  |**|  |30
  ------------------------
40|  |**|  |**|  |**|  |**|40
  ------------------------
50|**|  |**|  |**|  |**|  |50
  ------------------------
60|WP|WP|WP|WP|WP|WP|WP|WP|60
  ------------------------
70|WR|WN|WB|WK|WQ|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
CC CC CC
?22 22 22
?FF 22 26
?FF 22 63
?FF 26 34
?1F 63 43
?
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|BB|BK|BQ|BB|BN|BR|00
  ------------------------
10|BP|BP|BP|BP|BP|BP|BP|BP|10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|  |**|  |**|  |**|  |30
  ------------------------
40|  |**|  |WP|  |**|  |**|40
  ------------------------
50|**|  |**|  |**|  |**|  |50
  ------------------------
60|WP|WP|WP|**|WP|WP|WP|WP|60
  ------------------------
70|WR|WN|WB|WK|WQ|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 63 43
?.....................
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|BB|BK|BQ|BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|  |**|BP|**|  |**|  |30
  ------------------------
40|  |**|  |WP|  |**|  |**|40
  ------------------------
50|**|  |**|  |**|  |**|  |50
  ------------------------
60|WP|WP|WP|**|WP|WP|WP|WP|60
  ------------------------
70|WR|WN|WB|WK|WQ|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
0F 13 33
?0F 33 37
?0F 33 71
?FF 37 15
?16 71 52
?
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|BB|BK|BQ|BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|  |**|BP|**|  |**|  |30
  ------------------------
40|  |**|  |WP|  |**|  |**|40
  ------------------------
50|**|  |WN|  |**|  |**|  |50
  ------------------------
60|WP|WP|WP|**|WP|WP|WP|WP|60
  ------------------------
70|WR|  |WB|WK|WQ|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 71 52
?..........................
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|BB|BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|BQ|**|  |**|  |**|20
  ------------------------
30|**|  |**|BP|**|  |**|  |30
  ------------------------
40|  |**|  |WP|  |**|  |**|40
  ------------------------
50|**|  |WN|  |**|  |**|  |50
  ------------------------
60|WP|WP|WP|**|WP|WP|WP|WP|60
  ------------------------
70|WR|  |WB|WK|WQ|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
01 04 22
?FF 42 27
?01 22 72
?FF 27 23
?14 72 36
?
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|BB|BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|BQ|**|  |**|  |**|20
  ------------------------
30|**|  |**|BP|**|  |WB|  |30
  ------------------------
40|  |**|  |WP|  |**|  |**|40
  ------------------------
50|**|  |WN|  |**|  |**|  |50
  ------------------------
60|WP|WP|WP|**|WP|WP|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 72 36
?.................................
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|BQ|**|  |**|  |**|20
  ------------------------
30|**|  |**|BP|**|  |WB|  |30
  ------------------------
40|  |**|  |WP|  |**|BB|**|40
  ------------------------
50|**|  |WN|  |**|  |**|  |50
  ------------------------
60|WP|WP|WP|**|WP|WP|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
04 02 46
?FF 24 66
?04 46 65
?1B 66 54
?1D 65 45
?
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|BQ|**|  |**|  |**|20
  ------------------------
30|**|  |**|BP|**|  |WB|  |30
  ------------------------
40|  |**|  |WP|  |WP|BB|**|40
  ------------------------
50|**|  |WN|  |**|  |**|  |50
  ------------------------
60|WP|WP|WP|**|WP|**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 65 45
?FF 54 54
?1D 45 45
?FF 54 55
?1D 45 55
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|BQ|**|  |**|  |**|20
  ------------------------
30|**|  |**|BP|**|  |WB|  |30
  ------------------------
40|  |**|  |WP|  |**|BB|**|40
  ------------------------
50|**|  |WN|  |**|WP|**|  |50
  ------------------------
60|WP|WP|WP|**|WP|**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 45 55
?.....................................
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|BQ|**|  |**|  |**|20
  ------------------------
30|**|  |**|BP|**|BB|WB|  |30
  ------------------------
40|  |**|  |WP|  |**|  |**|40
  ------------------------
50|**|  |WN|  |**|WP|**|  |50
  ------------------------
60|WP|WP|WP|**|WP|**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
04 46 35
?FF 63 56
?04 35 64
?FF 56 44
?1E 64 44
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|BQ|**|  |**|  |**|20
  ------------------------
30|**|  |**|BP|**|BB|WB|  |30
  ------------------------
40|  |**|  |WP|WP|**|  |**|40
  ------------------------
50|**|  |WN|  |**|WP|**|  |50
  ------------------------
60|WP|WP|WP|**|  |**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 64 44
?....................................
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|BQ|**|  |**|  |**|20
  ------------------------
30|**|  |**|  |**|BB|WB|  |30
  ------------------------
40|  |**|  |WP|BP|**|  |**|40
  ------------------------
50|**|  |WN|  |**|WP|**|  |50
  ------------------------
60|WP|WP|WP|**|  |**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|WB|WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
0F 33 44
?FF 34 47
?0F 44 75
?FF 47 53
?15 75 31
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|BQ|**|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|BB|WB|  |30
  ------------------------
40|  |**|  |WP|BP|**|  |**|40
  ------------------------
50|**|  |WN|  |**|WP|**|  |50
  ------------------------
60|WP|WP|WP|**|  |**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|  |WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 75 31
?.....................................
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|  |BQ|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|BB|WB|  |30
  ------------------------
40|  |**|  |WP|BP|**|  |**|40
  ------------------------
50|**|  |WN|  |**|WP|**|  |50
  ------------------------
60|WP|WP|WP|**|  |**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|  |WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
01 22 23
?FF 22 35
?01 23 55
?04 35 54
?1D 55 44
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|  |BQ|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|BB|WB|  |30
  ------------------------
40|  |**|  |WP|WP|**|  |**|40
  ------------------------
50|**|  |WN|  |**|  |**|  |50
  ------------------------
60|WP|WP|WP|**|  |**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|  |WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 55 44
?.........................................
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|  |BQ|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|  |WB|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|**|  |WN|  |**|  |**|  |50
  ------------------------
60|WP|WP|WP|**|  |**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|  |WN|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
04 35 46
?FF 54 67
?04 46 76
?19 67 65
?17 76 55
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|  |BQ|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|  |WB|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|**|  |WN|  |**|WN|**|  |50
  ------------------------
60|WP|WP|WP|**|  |**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|  |**|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 76 55
?.........................................
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|  |WB|  |30
  ------------------------
40|  |BQ|  |WP|WP|**|BB|**|40
  ------------------------
50|**|  |WN|  |**|WN|**|  |50
  ------------------------
60|WP|WP|WP|**|  |**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|  |**|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
01 23 41
?FF 34 16
?01 41 60
?0B 16 05
?18 60 50
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|  |WB|  |30
  ------------------------
40|  |BQ|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|WN|**|  |50
  ------------------------
60|  |WP|WP|**|  |**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|  |**|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 60 50
?.....................................
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|  |WB|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|WN|**|  |50
  ------------------------
60|  |BQ|WP|**|  |**|WP|WP|60
  ------------------------
70|WR|  |**|WK|WQ|  |**|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
01 41 61
?0B 16 17
?01 61 73
?09 17 36
?10 73 63
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|BP|BP|BP|10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|  |WB|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|WN|**|  |50
  ------------------------
60|  |BQ|WP|WK|  |**|WP|WP|60
  ------------------------
70|WR|  |**|  |WQ|  |**|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 73 63
?.....................................
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|  |BP|BP|10
  ------------------------
20|  |**|  |**|  |BP|  |**|20
  ------------------------
30|**|WB|**|  |**|  |WB|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|WN|**|  |50
  ------------------------
60|  |BQ|WP|WK|  |**|WP|WP|60
  ------------------------
70|WR|  |**|  |WQ|  |**|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
0D 15 25
?16 52 57
?0D 25 74
?FF 57 47
?11 74 71
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|  |BP|BP|10
  ------------------------
20|  |**|  |**|  |BP|  |**|20
  ------------------------
30|**|WB|**|  |**|  |WB|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|WN|**|  |50
  ------------------------
60|  |BQ|WP|WK|  |**|WP|WP|60
  ------------------------
70|WR|WQ|**|  |**|  |**|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 74 71
?.....................................
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|  |BP|BP|10
  ------------------------
20|  |**|  |**|  |BP|  |**|20
  ------------------------
30|**|WB|**|  |**|  |WB|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|WN|**|  |50
  ------------------------
60|  |**|WP|WK|  |**|WP|WP|60
  ------------------------
70|BQ|WQ|**|  |**|  |**|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
01 61 70
?09 17 07
?01 70 71
?03 07 17
?11 71 70
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|  |BP|BP|10
  ------------------------
20|  |**|  |**|  |BP|  |**|20
  ------------------------
30|**|WB|**|  |**|  |WB|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|WN|**|  |50
  ------------------------
60|  |**|WP|WK|  |**|WP|WP|60
  ------------------------
70|WQ|  |**|  |**|  |**|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 71 70
?.........................
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|  |BP|BP|10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|  |BP|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|WN|**|  |50
  ------------------------
60|  |**|WP|WK|  |**|WP|WP|60
  ------------------------
70|WQ|  |**|  |**|  |**|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
0D 25 36
?FF 53 65
?0D 36 55
?FF 65 53
?17 55 36
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|  |BP|BP|10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|  |WN|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|  |**|  |50
  ------------------------
60|  |**|WP|WK|  |**|WP|WP|60
  ------------------------
70|WQ|  |**|  |**|  |**|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
FF 55 36
?.........................
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|  |BP|  |10
  ------------------------
20|  |**|  |**|  |**|  |BP|20
  ------------------------
30|**|WB|**|  |**|  |WN|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|  |**|  |50
  ------------------------
60|  |**|WP|WK|  |**|WP|WP|60
  ------------------------
70|WQ|  |**|  |**|  |**|WR|70
  ------------------------
   00 01 02 03 04 05 06 07
09 17 27
?FF 72 77
?09 27 77
?13 77 77
?13 77 75
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|  |BP|  |10
  ------------------------
20|  |**|  |**|  |**|  |BP|20
  ------------------------
30|**|WB|**|  |**|  |WN|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|  |**|  |50
  ------------------------
60|  |**|WP|WK|  |**|WP|WP|60
  ------------------------
70|WQ|  |**|  |**|WR|**|  |70
  ------------------------
   00 01 02 03 04 05 06 07
FF 77 75
?.........................
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |BB|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|  |BP|  |10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|  |BP|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|  |**|  |50
  ------------------------
60|  |**|WP|WK|  |**|WP|WP|60
  ------------------------
70|WQ|  |**|  |**|WR|**|  |70
  ------------------------
   00 01 02 03 04 05 06 07
09 27 36
?FF 73 67
?09 36 75
?19 67 50
?13 75 05
?
```

```
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |WR|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|  |BP|  |10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|  |BP|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|  |**|  |50
  ------------------------
60|  |**|WP|WK|  |**|WP|WP|60
  ------------------------
70|WQ|  |**|  |**|  |**|  |70
  ------------------------
   00 01 02 03 04 05 06 07
FF 75 05
?.............................
   00 01 02 03 04 05 06 07
  ------------------------
00|BR|BN|  |BK|  |WR|BN|BR|00
  ------------------------
10|BP|BP|BP|  |BP|  |BP|  |10
  ------------------------
20|  |**|  |**|  |**|  |**|20
  ------------------------
30|**|WB|**|  |**|  |BP|  |30
  ------------------------
40|  |**|  |WP|WP|**|BB|**|40
  ------------------------
50|WP|  |WN|  |**|  |**|  |50
  ------------------------
60|  |**|WP|WK|  |**|WP|WP|60
  ------------------------
70|WQ|  |**|  |**|  |**|  |70
  ------------------------
   00 01 02 03 04 05 06 07
FF FF FF
?
```

Appendix B – C Source Code for PICAXE 28X2

```
//*****************************************************************
//
//  Kim-1 MicroChess (c) 1976-2005 Peter Jennings, www.benlo.com
//  6502 emulation   (c) 2005 Bill Forster
//  28X2 emulation   (c) 2015 Ian Mitchell
//
//  Runs an emulation of the Kim-1 Microchess on the PICAXE 28X2
//  microcontroller. Based on an idea from Bill Forster to emulate
//  6502 microprocessor instructions in C. This program generates
//  the file 28X2Microchess.bas which can be uploaded to a 28X2.
//
//*****************************************************************

// All rights reserved.

// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
// 1. Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
// 2. Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
// 3. The name of the author may not be used to endorse or promote products
//    derived from this software without specific prior written permission.

// THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS OR
// IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
// OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
// IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
// NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES// LOSS OF USE,
// DATA, OR PROFITS// OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
// THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
// THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

#include <stdio.h>
#include <stdlib.h>

typedef unsigned char byte;
static FILE *f = NULL;
static int subnum = -1;

// 6502 emulation macros - register moves
#define T(src,dst) r();fprintf(f,"%s = %s : reg_f = %s\r\n",dst,src,src)
#define A "reg_a"
#define S "reg_s"
#define X "reg_x"
#define Y "reg_y"
#define TYA T(Y,A)
#define TAX T(A,X)
#define TAY T(A,Y)
#define TXA T(X,A)
#define TSX r();fprintf(f,"reg_x = ptr-0x100\r\n")
#define TXS r();fprintf(f,"ptr = reg_x+0x100\r\n")

// 6502 emulation macros - branches
```

```
#define BEQ(label) r();fprintf(f,"if reg_f=0 then goto %s\r\n",label)
#define BNE(label) r();fprintf(f,"if reg_f!=0 then goto %s\r\n",label)
#define BPL(label) r();fprintf(f,"if reg_f<0x80 then goto %s\r\n",label)
#define BMI(label) r();fprintf(f,"if reg_f>=0x80 then goto %s\r\n",label)
#define BCC(label) r();fprintf(f,"if reg_cy=0 then goto %s\r\n",label)
#define BCS(label) r();fprintf(f,"if reg_cy!=0 then goto %s\r\n",label)
#define BVC(label) r();fprintf(f,"if reg_v=0 then goto %s\r\n",label)
#define BVS(label) r();fprintf(f,"if reg_v!=0 then goto %s\r\n",label)
#define BRA(label) r();fprintf(f,"goto %s\r\n",label)
#define JEQ(label) r();fprintf(f,"if reg_f=0 then goto %s\r\n",label)
#define JMP(label) r();fprintf(f,"goto %s\r\n",label)
#define JSR(func)  r();fprintf(f,"@ptrdec = %d : goto %s\r\n",++subnum,func)
#define JSR_(func) r();fprintf(f,"gosub %s\r\n",func)
#define RTS        r();fprintf(f,"goto __return__\r\n")
#define RTS_       r();fprintf(f,"return\r\n")

// 6502 emulation macros - load registers
//  Addressing conventions;
//   default addressing mode is zero page, else indicate with suffix;
//   i = immediate
//   x = indexed, zero page
//   f = indexed, not zero page (f for "far")
#define LDAi(dat8)       r();fprintf(f,"reg_a = 0x%02x : reg_f = reg_a\r\n",dat8)
#define LDAi_(dat8)      r();fprintf(f,"reg_a = 0x%02x\r\n",dat8)
#define LDAx(addr8,idx)  r();fprintf(f,"reg_f = %s+0x%02x : get reg_f,reg_a :
reg_f = reg_a\r\n",idx,addr8)
#define LDAf(addr16,idx) r();fprintf(f,"twobytes = %s+0x%02x : get twobytes,reg_a
: reg_f = reg_a\r\n",idx,addr16)
#define LDA(addr8)       r();fprintf(f,"get 0x%02x,reg_a : reg_f =
reg_a\r\n",addr8)
#define LDA_(addr8)      r();fprintf(f,"get 0x%02x,reg_a\r\n",addr8)
#define LDXi(dat8)       r();fprintf(f,"reg_x = 0x%02x : reg_f = reg_x\r\n",dat8)
#define LDXi_(dat8)      r();fprintf(f,"reg_x = 0x%02x\r\n",dat8)
#define LDX(addr8)       r();fprintf(f,"get 0x%02x,reg_x : reg_f =
reg_x\r\n",addr8)
#define LDYi(dat8)       r();fprintf(f,"reg_y = 0x%02x : reg_f = reg_y\r\n",dat8)
#define LDY(addr8)       r();fprintf(f,"get 0x%02x,reg_y : reg_f =
reg_y\r\n",addr8)
#define LDYx(addr8,idx)  r();fprintf(f,"reg_f = %s+0x%02x : get reg_f,reg_y :
reg_f = reg_y\r\n",idx,addr8)
#define LDYx_(addr8,idx) r();fprintf(f,"reg_f = %s+0x%02x : get
reg_f,reg_y\r\n",idx,addr8)

// 6502 emulation macros - store registers
#define STA(addr8)       r();fprintf(f,"put 0x%02x,reg_a\r\n",addr8)
#define STAx(addr8,idx)  r();fprintf(f,"temp = %s+0x%02x : put
temp,reg_a\r\n",idx,addr8)
#define STX(addr8)       r();fprintf(f,"put 0x%02x,reg_x\r\n",addr8)
#define STY(addr8)       r();fprintf(f,"put 0x%02x,reg_y\r\n",addr8)
#define STYx(addr8,idx)  r();fprintf(f,"temp = %s+0x%02x : put
temp,reg_y\r\n",idx,addr8)

// 6502 emulation macros - set/clear flags
#define CLD              // luckily CPU's BCD flag is cleared then never set
#define CLC              r();fprintf(f,"reg_cy = 0\r\n");
#define SEC              r();fprintf(f,"reg_cy = 1\r\n");
#define CLV              r();fprintf(f,"reg_v = 0\r\n");
#define SEV /*extra*/    r();fprintf(f,"reg_v = 1\r\n"); /*avoid problematic V
emulation*/
```

```
// 6502 emulation macros - accumulator logical operations
#define ANDi(dat8)       r();fprintf(f,"reg_a = reg_a&0x%02x : reg_f =
reg_a\r\n",dat8)
#define ANDi_(dat8)      r();fprintf(f,"reg_a = reg_a&0x%02x\r\n",dat8)
#define ORA(addr8)       r();fprintf(f,"get 0x%02x,reg_f : reg_a = reg_a|reg_f :
reg_f = reg_a\r\n",addr8)
#define ORA_(addr8)      r();fprintf(f,"get 0x%02x,reg_f : reg_a =
reg_a|reg_f\r\n",addr8)

// 6502 emulation macros - shifts and rotates
#define ASL(addr8)       r();fprintf(f,"get 0x%02x,reg_f : reg_cy = reg_f>>7 :
reg_f = reg_f<<1 : put 0x%02x,reg_f\r\n",addr8,addr8)
#define ROL(addr8)       r();fprintf(f,"get 0x%02x,reg_f : temp = reg_f>>7 : reg_f
= reg_f<<1|reg_cy : put 0x%02x,reg_f : reg_cy = temp\r\n",addr8,addr8)
#define LSR              r();fprintf(f,"reg_cy = reg_a&0x01 : reg_a = reg_a>>1 :
reg_f = reg_a\r\n")

// 6502 emulation macros - push and pull
#define PHA              r();fprintf(f,"@ptrdec = reg_a\r\n")
#define PLA              r();fprintf(f,"inc ptr : reg_a = @ptr\r\n")
#define PHY              r();fprintf(f,"@ptrdec = reg_y\r\n")
#define PLY              r();fprintf(f,"inc ptr : reg_y = @ptr\r\n")
#define PHP              r();fprintf(f,"temp = reg_v<<1|reg_cy : temp =
reg_f>>7<<3|temp : temp = reg_f max 1<<2|temp : @ptrdec = temp\r\n")
#define PLP              r();fprintf(f,"inc ptr : temp = @ptr : reg_f =
temp<<4&0xc0: reg_cy = temp&0x01 : reg_v = temp>>1&0x01\r\n")

// 6502 emulation macros - compare
// use ones complement plus one to get the correct difference and carry out
#define CMPi(dat8)       r();fprintf(f,"reg_fc = reg_a+0x%02x ;
CMPi(0x%02x)\r\n",(~(dat8)+1)&0xff,dat8)
#define CMP(addr8)       r();fprintf(f,"get 0x%02x,reg_f : reg_f = not reg_f :
reg_fc = reg_a+reg_f+1\r\n",addr8)
#define CMPx(addr8,idx)  r();fprintf(f,"reg_f = %s+0x%02x : get reg_f,reg_f :
reg_f = not reg_f : reg_fc = reg_a+reg_f+1\r\n",idx,addr8)
#define CMPf(addr16,idx) r();fprintf(f,"twobytes = %s+0x%02x : get twobytes,reg_f
: reg_f = not reg_f : reg_fc = reg_a+reg_f+1\r\n",idx,addr16)
#define CPXi(dat8)       r();fprintf(f,"reg_fc = reg_x+0x%02x ;
CPXi(0x%02x)\r\n",(~(dat8)+1)&0xff,dat8)
#define CPXf(addr16,idx) r();fprintf(f,"twobytes = %s+0x%02x : get twobytes,reg_f
: reg_f = not reg_f : reg_fc = reg_x+reg_f+1\r\n",idx,addr16)
#define CPYi(dat8)       r();fprintf(f,"reg_fc = reg_y+0x%02x ;
CPYi(0x%02x)\r\n",(~(dat8)+1)&0xff,dat8)

// 6502 emulation macros - increment,decrement
#define DEX              r();fprintf(f,"dec reg_x : reg_f = reg_x\r\n")
#define DEY              r();fprintf(f,"dec reg_y : reg_f = reg_y\r\n")
#define DEC(addr8)       r();fprintf(f,"get 0x%02x,reg_f : dec reg_f : put
0x%02x,reg_f\r\n",addr8,addr8)
#define INX              r();fprintf(f,"inc reg_x : reg_f = reg_x\r\n")
#define INY              r();fprintf(f,"inc reg_y : reg_f = reg_y\r\n")
#define INC(addr8)       r();fprintf(f,"get 0x%02x,reg_f : inc reg_f : put
0x%02x,reg_f\r\n",addr8,addr8)
#define INCx(addr8,idx)  r();fprintf(f,"temp = %s+0x%02x : get temp,reg_f : inc
reg_f : put temp,reg_f\r\n",idx,addr8)

// 6502 emulation macros - add
```

```
#define ADCi(dat8)       r();fprintf(f,"reg_fc = reg_a+0x%02x+reg_cy : reg_a =
reg_f\r\n",dat8)
#define ADC(addr8)       r();fprintf(f,"get 0x%02x,reg_f : reg_fc =
reg_a+reg_f+reg_cy : reg_a = reg_f\r\n",addr8)
#define ADCx(addr8,idx)  r();fprintf(f,"reg_f = %s+0x%02x : get reg_f,reg_f :
reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f\r\n",idx,addr8)
#define ADCf(addr16,idx) r();fprintf(f,"twobytes = %s+0x%02x : get twobytes,reg_f
: reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f\r\n",idx,addr16)

// 6502 emulation macros - subtraction
//   (note that using ones complement both as an input and an output
//   the carry flag has opposite sense to that used for adc)
#define SBC(addr8)       r();fprintf(f,"get 0x%02x,reg_f : reg_f = not reg_f :
reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f\r\n",addr8)
#define SBCx(addr8,idx)  r();fprintf(f,"reg_f = %s+0x%02x : get reg_f,reg_f :
reg_f = not reg_f: reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f\r\n",idx,addr8)

// page zero variables
static const byte BOARD  = 0x50;
static const byte BK     = 0x60;
static const byte PIECE  = 0xB0;
static const byte SQUARE = 0xB1;
static const byte SP2    = 0xB2;
static const byte SP1    = 0xB3;
static const byte INCHEK = 0xB4;
static const byte STATE  = 0xB5;
static const byte MOVEN  = 0xB6;
static const byte REV    = 0xB7;
static const byte OMOVE  = 0xDC;
static const byte WCAP0  = 0xDD;
static const byte COUNT  = 0xDE;
static const byte BCAP2  = 0xDE;
static const byte WCAP2  = 0xDF;
static const byte BCAP1  = 0xE0;
static const byte WCAP1  = 0xE1;
static const byte BCAP0  = 0xE2;
static const byte MOB    = 0xE3;
static const byte MAXC   = 0xE4;
static const byte CC     = 0xE5;
static const byte PCAP   = 0xE6;
static const byte BMOB   = 0xE3;
static const byte BMAXC  = 0xE4;
static const byte BMCC   = 0xE5;     // was BCC, make sure not confused with
instruction definition
static const byte BMAXP  = 0xE6;
static const byte XMAXC  = 0xE8;
static const byte WMOB   = 0xEB;
static const byte WMAXC  = 0xEC;
static const byte WCC    = 0xED;
static const byte WMAXP  = 0xEE;
static const byte PMOB   = 0xEF;
static const byte PMAXC  = 0xF0;
static const byte PCC    = 0xF1;
static const byte PCP    = 0xF2;
static const byte OLDKY  = 0xF3;
static const byte BESTP  = 0xFB;
static const byte BESTV  = 0xFA;
static const byte BESTM  = 0xF9;
static const byte DIS1   = 0xFB;
```

```
static const byte DIS2   = 0xFA;
static const byte DIS3   = 0xF9;
static const byte temp   = 0xFC;

static const byte SETW_data[] =
{
   0x03, 0x04, 0x00, 0x07, 0x02, 0x05, 0x01, 0x06,
   0x10, 0x17, 0x11, 0x16, 0x12, 0x15, 0x14, 0x13,
   0x73, 0x74, 0x70, 0x77, 0x72, 0x75, 0x71, 0x76,
   0x60, 0x67, 0x61, 0x66, 0x62, 0x65, 0x64, 0x63
};
static const byte MOVEX_data[]  =
{
   0x00, 0xF0, 0xFF, 0x01, 0x10, 0x11, 0x0F, 0xEF, 0xF1,
   0xDF, 0xE1, 0xEE, 0xF2, 0x12, 0x0E, 0x1F, 0x21
};
static const byte POINTS_data[] =
{
   0x0B, 0x0A, 0x06, 0x06, 0x04, 0x04, 0x04, 0x04,
   0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02
};
static const byte OPNING_data[] =
{
   0x99, 0x25, 0x0B, 0x25, 0x01, 0x00, 0x33, 0x25,
   0x07, 0x36, 0x34, 0x0D, 0x34, 0x34, 0x0E, 0x52,
   0x25, 0x0D, 0x45, 0x35, 0x04, 0x55, 0x22, 0x06,
   0x43, 0x33, 0x0F, 0xCC
};
static const unsigned int SETW = 0x200;
static const unsigned int MOVEX = sizeof(SETW_data)+SETW;
static const unsigned int POINTS = sizeof(MOVEX_data)+MOVEX;
static const unsigned int OPNING = sizeof(POINTS_data)+POINTS;

static const unsigned int LEVEL1 = 0x00;
static const unsigned int LEVEL2 = 0x01;

// label destinations
#define RESTART_CHESS_ fprintf(f,"_RESTART_CHESS_:\r\n");
#define CHESS_BEGIN_   fprintf(f,"_CHESS_BEGIN_:\r\n");
#define INITCLEAR_     fprintf(f,"_INITCLEAR_:\r\n");
#define WHSET_         fprintf(f,"_WHSET_:\r\n");
#define NOSET_         fprintf(f,"_NOSET_:\r\n");
#define NOREV_         fprintf(f,"_NOREV_:\r\n");
#define CLDSP_         fprintf(f,"_CLDSP_:\r\n");
#define CLDSP2_        fprintf(f,"_CLDSP2_:\r\n");
#define NOGO_          fprintf(f,"_NOGO_:\r\n");
#define NOMV_          fprintf(f,"_NOMV_:\r\n");
#define DONE_          fprintf(f,"_DONE_:\r\n");
#define JANUS_         fprintf(f,"_JANUS_:\r\n");
#define OVER_          fprintf(f,"_OVER_:\r\n");
#define NOQ_           fprintf(f,"_NOQ_:\r\n");
#define ELOOP_         fprintf(f,"_ELOOP_:\r\n");
#define FOUN_          fprintf(f,"_FOUN_:\r\n");
#define LESS_          fprintf(f,"_LESS_:\r\n");
#define NOCAP_         fprintf(f,"_NOCAP_:\r\n");
#define XRT_           fprintf(f,"_XRT_:\r\n");
#define ON4_           fprintf(f,"_ON4_:\r\n");
#define NOCOUNT_       fprintf(f,"_NOCOUNT_:\r\n");
#define RETJ_          fprintf(f,"_RETJ_:\r\n");
```

```
#define TREE_          fprintf(f,"_TREE_:\r\n");
#define LOOPX_         fprintf(f,"_LOOPX_:\r\n");
#define FOUNX_         fprintf(f,"_FOUNX_:\r\n");
#define NOMAX_         fprintf(f,"_NOMAX_:\r\n");
#define UPTREE_        fprintf(f,"_UPTREE_:\r\n");
#define INPUT_         fprintf(f,"_INPUT_:\r\n");
#define ERROR_         fprintf(f,"_ERROR_:\r\n");
#define DISP_          fprintf(f,"_DISP_:\r\n");
#define SEARCH_        fprintf(f,"_SEARCH_:\r\n");
#define HERE_          fprintf(f,"_HERE_:\r\n");
#define GNMZ_          fprintf(f,"_GNMZ_:\r\n");
#define GNMX_          fprintf(f,"_GNMX_:\r\n");
#define CLEAR_         fprintf(f,"_CLEAR_:\r\n");
#define GNM_           fprintf(f,"_GNM_:\r\n");
#define NEWP_          fprintf(f,"_NEWP_:\r\n");
#define NEX_           fprintf(f,"_NEX_:\r\n");
#define KING_          fprintf(f,"_KING_:\r\n");
#define QUEEN_         fprintf(f,"_QUEEN_:\r\n");
#define ROOK_          fprintf(f,"_ROOK_:\r\n");
#define AGNR_          fprintf(f,"_AGNR_:\r\n");
#define BISHOP_        fprintf(f,"_BISHOP_:\r\n");
#define KNIGHT_        fprintf(f,"_KNIGHT_:\r\n");
#define AGNN_          fprintf(f,"_AGNN_:\r\n");
#define PAWN_          fprintf(f,"_PAWN_:\r\n");
#define P1_            fprintf(f,"_P1_:\r\n");
#define P2_            fprintf(f,"_P2_:\r\n");
#define P3_            fprintf(f,"_P3_:\r\n");
#define SNGMV_         fprintf(f,"_SNGMV_:\r\n");
#define ILL1_          fprintf(f,"_ILL1_:\r\n");
#define LINE_          fprintf(f,"_LINE_:\r\n");
#define OVL_           fprintf(f,"_OVL_:\r\n");
#define ILL_           fprintf(f,"_ILL_:\r\n");
#define REVERSE_       fprintf(f,"_REVERSE_:\r\n");
#define ETC_           fprintf(f,"_ETC_:\r\n");
#define CMOVE_         fprintf(f,"_CMOVE_:\r\n");
#define LOOP_          fprintf(f,"_LOOP_:\r\n");
#define NO_            fprintf(f,"_NO_:\r\n");
#define SPX_           fprintf(f,"_SPX_:\r\n");
#define RETL_          fprintf(f,"_RETL_:\r\n");
#define ILLEGAL_       fprintf(f,"_ILLEGAL_:\r\n");
#define RESET_         fprintf(f,"_RESET_:\r\n");
#define GENRM_         fprintf(f,"_GENRM_:\r\n");
#define RUM_           fprintf(f,"_RUM_:\r\n");
#define UMOVE_         fprintf(f,"_UMOVE_:\r\n");
#define MOVE_          fprintf(f,"_MOVE_:\r\n");
#define CHECK_         fprintf(f,"_CHECK_:\r\n");
#define TAKE_          fprintf(f,"_TAKE_:\r\n");
#define STRV_          fprintf(f,"_STRV_:\r\n");
#define CKMATE_        fprintf(f,"_CKMATE_:\r\n");
#define NOCHEK_        fprintf(f,"_NOCHEK_:\r\n");
#define RETV_          fprintf(f,"_RETV_:\r\n");
#define RETP_          fprintf(f,"_RETP_:\r\n");
#define GO_            fprintf(f,"_GO_:\r\n");
#define END_           fprintf(f,"_END_:\r\n");
#define NOOPEN_        fprintf(f,"_NOOPEN_:\r\n");
#define MV2_           fprintf(f,"_MV2_:\r\n");
#define MATE_          fprintf(f,"_MATE_:\r\n");
#define DISMV_         fprintf(f,"_DISMV_:\r\n");
#define DROL_          fprintf(f,"_DROL_:\r\n");
```

```
#define STRATGY_       fprintf(f,"_STRATGY_:\r\n");
#define POS_           fprintf(f,"_POS_:\r\n");
#define POSN_          fprintf(f,"_POSN_:\r\n");
#define NOPOSN_        fprintf(f,"_NOPOSN_:\r\n");
#define POUT_          fprintf(f,"_POUT_:\r\n");
#define KIN_           fprintf(f,"_KIN_:\r\n");
#define SETUP_         fprintf(f,"_SETUP_:\r\n");
#define TESTLEVEL2_    fprintf(f,"_TESTLEVEL2_:\r\n");
#define TESTLEVEL3_    fprintf(f,"_TESTLEVEL3_:\r\n");
#define TESTSAVE_      fprintf(f,"_TESTSAVE_:\r\n");
#define TESTLOAD_      fprintf(f,"_TESTLOAD_:\r\n");
#define TESTRESTORE_   fprintf(f,"_TESTRESTORE_:\r\n");
#define TESTUNDO_      fprintf(f,"_TESTUNDO_:\r\n");

// labels
#define RESTART_CHESS  "_RESTART_CHESS_"
#define CHESS_BEGIN    "_CHESS_BEGIN_"
#define INITCLEAR      "_INITCLEAR_"
#define WHSET          "_WHSET_"
#define NOSET          "_NOSET_"
#define NOREV          "_NOREV_"
#define CLDSP          "_CLDSP_"
#define CLDSP2         "_CLDSP2_"
#define NOGO           "_NOGO_"
#define NOMV           "_NOMV_"
#define DONE           "_DONE_"
#define JANUS          "_JANUS_"
#define OVER           "_OVER_"
#define NOQ            "_NOQ_"
#define ELOOP          "_ELOOP_"
#define FOUN           "_FOUN_"
#define LESS           "_LESS_"
#define NOCAP          "_NOCAP_"
#define XRT            "_XRT_"
#define ON4            "_ON4_"
#define NOCOUNT        "_NOCOUNT_"
#define RETJ           "_RETJ_"
#define TREE           "_TREE_"
#define LOOPX          "_LOOPX_"
#define FOUNX          "_FOUNX_"
#define NOMAX          "_NOMAX_"
#define UPTREE         "_UPTREE_"
#define INPUT          "_INPUT_"
#define ERROR          "_ERROR_"
#define DISP           "_DISP_"
#define SEARCH         "_SEARCH_"
#define HERE           "_HERE_"
#define GNMZ           "_GNMZ_"
#define GNMX           "_GNMX_"
#define CLEAR          "_CLEAR_"
#define GNM            "_GNM_"
#define NEWP           "_NEWP_"
#define NEX            "_NEX_"
#define KING           "_KING_"
#define QUEEN          "_QUEEN_"
#define ROOK           "_ROOK_"
#define AGNR           "_AGNR_"
#define BISHOP         "_BISHOP_"
#define KNIGHT         "_KNIGHT_"
```

```
#define AGNN           "_AGNN_"
#define PAWN           "_PAWN_"
#define P1             "_P1_"
#define P2             "_P2_"
#define P3             "_P3_"
#define SNGMV          "_SNGMV_"
#define ILL1           "_ILL1_"
#define LINE           "_LINE_"
#define OVL            "_OVL_"
#define ILL            "_ILL_"
#define REVERSE        "_REVERSE_"
#define ETC            "_ETC_"
#define CMOVE          "_CMOVE_"
#define LOOP           "_LOOP_"
#define NO             "_NO_"
#define SPX            "_SPX_"
#define RETL           "_RETL_"
#define ILLEGAL        "_ILLEGAL_"
#define RESET          "_RESET_"
#define GENRM          "_GENRM_"
#define RUM            "_RUM_"
#define UMOVE          "_UMOVE_"
#define MOVE           "_MOVE_"
#define CHECK          "_CHECK_"
#define TAKE           "_TAKE_"
#define STRV           "_STRV_"
#define CKMATE         "_CKMATE_"
#define NOCHEK         "_NOCHEK_"
#define RETV           "_RETV_"
#define RETP           "_RETP_"
#define GO             "_GO_"
#define END            "_END_"
#define NOOPEN         "_NOOPEN_"
#define MV2            "_MV2_"
#define MATE           "_MATE_"
#define DISMV          "_DISMV_"
#define DROL           "_DROL_"
#define STRATGY        "_STRATGY_"
#define POS            "_POS_"
#define POSN           "_POSN_"
#define NOPOSN         "_NOPOSN_"
#define POUT           "_POUT_"
#define KIN            "_KIN_"
#define SETUP          "_SETUP_"
#define TESTLEVEL2     "_TESTLEVEL2_"
#define TESTLEVEL3     "_TESTLEVEL3_"
#define TESTSAVE       "_TESTSAVE_"
#define TESTLOAD       "_TESTLOAD_"
#define TESTRESTORE    "_TESTRESTORE_"
#define TESTUNDO       "_TESTUNDO_"

static void r(void)
{
   static int testsubnum = -1;
   if (subnum==testsubnum)
   {
      fprintf(f,"      ");
      return;
   }
```

```
   testsubnum = subnum;
   fprintf(f,"_%02d:  ",subnum);
}

static void copyright(void)
{
   fprintf(f,"#rem\r\n");

fprintf(f,"*************************************************************************
\r\n");

fprintf(f,"*************************************************************************
\r\n");

fprintf(f,"*************************************************************************
\r\n");

fprintf(f,"*************************************************************************
\r\n");

fprintf(f,"*************************************************************************
\r\n");
   fprintf(f,"\r\n");
   fprintf(f,"  Kim-1 MicroChess (c) 1976-2005 Peter Jennings,
www.benlo.com\r\n");
   fprintf(f,"  6502 emulation   (c) 2005 Bill Forster\r\n");
   fprintf(f,"  28X2 emulation   (c) 2015 Ian Mitchell\r\n");
   fprintf(f,"\r\n");
   fprintf(f,"  Runs an emulation of the Kim-1 Microchess on the PICAXE
28X2\r\n");
   fprintf(f,"  microcontroller. Based on an idea from Bill Forster to
emulate\r\n");
   fprintf(f,"  6502 microprocessor instructions in C. The program is
created\r\n");
   fprintf(f,"  by running 28X2Microchess.exe. This file
(28X2Microchess.bas)\r\n");
   fprintf(f,"  is generated and can be uploaded to a 28X2.\r\n");
   fprintf(f,"\r\n");

fprintf(f,"*************************************************************************
\r\n");

fprintf(f,"*************************************************************************
\r\n");

fprintf(f,"*************************************************************************
\r\n");

fprintf(f,"*************************************************************************
\r\n");

fprintf(f,"*************************************************************************
\r\n");
   fprintf(f,"\r\n");
   fprintf(f," All rights reserved.\r\n");
   fprintf(f,"\r\n");
   fprintf(f," Redistribution and use in source and binary forms, with or
without\r\n");
```

```
   fprintf(f," modification, are permitted provided that the following
conditions\r\n");
   fprintf(f," are met:\r\n");
   fprintf(f," 1. Redistributions of source code must retain the above
copyright\r\n");
   fprintf(f,"       notice, this list of conditions and the following
disclaimer.\r\n");
   fprintf(f," 2. Redistributions in binary form must reproduce the above
copyright\r\n");
   fprintf(f,"       notice, this list of conditions and the following disclaimer
in the\r\n");
   fprintf(f,"       documentation and/or other materials provided with the
distribution.\r\n");
   fprintf(f," 3. The name of the author may not be used to endorse or promote
products\r\n");
   fprintf(f,"       derived from this software without specific prior written
permission.\r\n");
   fprintf(f,"\r\n");
   fprintf(f," THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS
OR\r\n");
   fprintf(f," IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES\r\n");
   fprintf(f," OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED.\r\n");
   fprintf(f," IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT,
INDIRECT,\r\n");
   fprintf(f," INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT\r\n");
   fprintf(f," NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES LOSS OF
USE,\r\n");
   fprintf(f," DATA, OR PROFITS OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY\r\n");
   fprintf(f," THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT\r\n");
   fprintf(f," (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF\r\n");
   fprintf(f," THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.\r\n");
   fprintf(f,"#endrem\r\n");
   fprintf(f,"\r\n");
}

static void init(void)
{
//   errno_t e = fopen_s(&f,"28X2Microchess.bas","wb");
//   if (e!=0)
   f = fopen("28X2Microchess.bas","wb");
   if (f==NULL)
   {
      printf("could not open file\r\n");
      exit(1);
   }
   copyright();
   fprintf(f,"#picaxe 28x2\r\n");
   fprintf(f,"setfreq em64\r\n");
   fprintf(f,";setfreq m16\r\n");
   fprintf(f,"symbol twobytes = w0\r\n");
   fprintf(f,"symbol reg_a = b2\r\n");
   fprintf(f,"symbol reg_x = b3\r\n");
```

```
   fprintf(f,"symbol reg_y = b4\r\n");
   fprintf(f,"symbol reg_v = b5\r\n");
   fprintf(f,"symbol reg_f = b6\r\n");
   fprintf(f,"symbol reg_cy = b7\r\n");
   fprintf(f,"symbol reg_fc = w3\r\n");
   fprintf(f,"symbol temp = b8\r\n");
   fprintf(f,"\r\n");
   fprintf(f,"symbol _row = b9\r\n");
   fprintf(f,"symbol _col = b10\r\n");
   fprintf(f,"symbol _loc = b11\r\n");
   fprintf(f,"symbol _pindex = b12\r\n");
   fprintf(f,"symbol _reverse = b13\r\n");
   fprintf(f,"symbol _p = b14\r\n");
   fprintf(f,"\r\n");
   fprintf(f,"      gosub __read_static_data__\r\n");
}

static void done(void)
{
   fprintf(f,"__hexbyte__:\r\n");
   fprintf(f,"      temp = reg_a>>4+\"0\"\r\n");
   fprintf(f,"      gosub __nybble__\r\n");
   fprintf(f,"      temp = reg_a&0x0f+\"0\"\r\n");
   fprintf(f,"__nybble__:\r\n");
   fprintf(f,"      if temp>\"9\" then : temp = temp+7 : endif\r\n");
   fprintf(f,"      sertxd (temp)\r\n");
   fprintf(f,"      return\r\n");
   fprintf(f,"__showboard__:\r\n");
   fprintf(f,"      gosub __backupposition__\r\n");
   fprintf(f,"      sertxd (cr,lf)\r\n");
   fprintf(f,"      gosub __rownum__\r\n");
   fprintf(f,"      gosub __line__\r\n");
   fprintf(f,"      get 0xb7,_reverse\r\n");
   fprintf(f,"      for _row=0 to 7\r\n");
   fprintf(f,"         sertxd (#_row,\"0|\")\r\n");
   fprintf(f,"         for _col = 0 to 7\r\n");
   fprintf(f,"            _loc = _row<<4+_col\r\n");
   fprintf(f,"            for _pindex = 0 to 0x1f\r\n");
   fprintf(f,"               _p = _pindex+0x50\r\n");
   fprintf(f,"               get _p,_p\r\n");
   fprintf(f,"               if _p=_loc then\r\n");
   fprintf(f,"                  _p = _pindex>>4^_reverse\r\n");
   fprintf(f,"                  if _p=0 then sertxd (\"W\") : else : sertxd
(\"B\") : endif\r\n");
   fprintf(f,"                  _p = _pindex&0x0f\r\n");
   fprintf(f,"                  lookup _p,(\"KQRRBBNNPPPPPPPP\"),_p\r\n");
   fprintf(f,"                  sertxd (_p)\r\n");
   fprintf(f,"                  goto __next_location__\r\n");
   fprintf(f,"               endif\r\n");
   fprintf(f,"            next\r\n");
   fprintf(f,"            ; not found\r\n");
   fprintf(f,"            _p = _row^_col&1\r\n");
   fprintf(f,"            if _p=1 then : sertxd (\"**\") : else : sertxd (\"  \")
: endif\r\n");
   fprintf(f,"            __next_location__:\r\n");
   fprintf(f,"            sertxd (\"|\")\r\n");
   fprintf(f,"         next\r\n");
   fprintf(f,"         sertxd (#_row,\"0\",cr,lf)\r\n");
   fprintf(f,"         gosub __line__\r\n");
```

```
   fprintf(f,"      next\r\n");
   fprintf(f,"      gosub __rownum__\r\n");
   fprintf(f,"      return\r\n");
   fprintf(f,"__line__:\r\n");
   fprintf(f,"      sertxd (\"  \") : for _p=1 to 25 : sertxd (\"-\") : next :
sertxd (cr,lf)\r\n");
   fprintf(f,"      return\r\n");
   fprintf(f,"__rownum__:\r\n");
   fprintf(f,"      sertxd (\"  \") : for _p=0 to 7 : sertxd (\" 0\",#_p) : next :
sertxd (cr,lf)\r\n");
   fprintf(f,"      return\r\n");

   // emulate RTS
   fprintf(f,"__return__:\r\n");
   fprintf(f,"      inc ptr\r\n");
   fprintf(f,"      branch @ptr,(");
   for (int i=0;i<=subnum;i++)
   {
      if (i>0)
      {
         fprintf(f,",");
      }
      fprintf(f,"_%02d",i);
   }
   fprintf(f,")\r\n");

   // save board
   fprintf(f,"__saveposition__:\r\n");
   fprintf(f,"      write 0xff,0xff\r\n");
   fprintf(f,"      _loc = 0x64\r\n");

   // save position
   fprintf(f,"__saveposition0__:\r\n");
   fprintf(f,"      ;_loc has eeprom address\r\n");
   fprintf(f,"      for temp = 0 to 0x1f\r\n");
   fprintf(f,"         _p = temp+0x%02x\r\n",BOARD);
   fprintf(f,"         get _p,_p\r\n");
   fprintf(f,"         _pindex =_loc+temp\r\n");
   fprintf(f,"         write _pindex,_p\r\n");
   fprintf(f,"      next\r\n");
   fprintf(f,"      get 0x%02x,_p\r\n",REV);
   fprintf(f,"      inc _pindex\r\n");
   fprintf(f,"      write _pindex,_p\r\n");
   fprintf(f,"      get 0x%02x,_p\r\n",LEVEL1);
   fprintf(f,"      inc _pindex\r\n");
   fprintf(f,"      write _pindex,_p\r\n");
   fprintf(f,"      get 0x%02x,_p\r\n",LEVEL2);
   fprintf(f,"      inc _pindex\r\n");
   fprintf(f,"      write _pindex,_p\r\n");
   fprintf(f,"      return\r\n");

   // load position
   fprintf(f,"__loadposition0__:\r\n");
   fprintf(f,"      ;_loc has eeprom address\r\n");
   fprintf(f,"      for temp = 0 to 0x1f\r\n");
   fprintf(f,"         _pindex =_loc+temp\r\n");
   fprintf(f,"         read _pindex,_p\r\n");
   fprintf(f,"         _pindex = temp+0x%02x\r\n",BOARD);
   fprintf(f,"         put _pindex,_p\r\n");
```

```
fprintf(f,"      next\r\n");
fprintf(f,"      _pindex = _loc+0x20\r\n");
fprintf(f,"      read _pindex,_p\r\n");
fprintf(f,"      put 0x%02x,_p\r\n",REV);
fprintf(f,"      inc _pindex\r\n");
fprintf(f,"      read _pindex,_p\r\n");
fprintf(f,"      put 0x%02x,_p\r\n",LEVEL1);
fprintf(f,"      inc _pindex\r\n");
fprintf(f,"      read _pindex,_p\r\n");
fprintf(f,"      put 0x%02x,_p\r\n",LEVEL2);
fprintf(f,"      return\r\n");

// load board
fprintf(f,"__loadposition__:\r\n");
fprintf(f,"      read 0xff,temp\r\n");
fprintf(f,"      if temp!=0xff then\r\n");
fprintf(f,"         sertxd (cr,lf,\"Save first.\",cr,lf)\r\n");
fprintf(f,"         return\r\n");
fprintf(f,"      endif\r\n");
fprintf(f,"      _loc = 0x64\r\n");
fprintf(f,"      gosub __loadposition0__\r\n");
fprintf(f,"      goto __showboard__\r\n");

// backup called after every move
fprintf(f,"__backupposition__:\r\n");
fprintf(f,"      write 0xfe,0xff\r\n");
fprintf(f,"      _loc = 0x8c\r\n");
fprintf(f,"      goto __saveposition0__\r\n");

// restore position
fprintf(f,"__restoreposition__:\r\n");
fprintf(f,"      read 0xfe,temp\r\n");
fprintf(f,"      if temp!=0xff then\r\n");
fprintf(f,"         sertxd (cr,lf,\"Can't restore.\",cr,lf)\r\n");
fprintf(f,"         return\r\n");
fprintf(f,"      endif\r\n");
fprintf(f,"      _loc = 0x8c\r\n");
fprintf(f,"      gosub __loadposition0__\r\n");
fprintf(f,"      goto __showboard__\r\n");

// called after every computer move
fprintf(f,"__dosaveposition__:\r\n");
fprintf(f,"      write 0xfd,0xff\r\n");
fprintf(f,"      _loc = 0xb4\r\n");
fprintf(f,"      goto __saveposition0__\r\n");

// undo position
fprintf(f,"__undoposition__:\r\n");
fprintf(f,"      read 0xfd,temp\r\n");
fprintf(f,"      if temp!=0xff then\r\n");
fprintf(f,"         sertxd (cr,lf,\"Can't undo.\",cr,lf)\r\n");
fprintf(f,"         return\r\n");
fprintf(f,"      endif\r\n");
fprintf(f,"      _loc = 0xb4\r\n");
fprintf(f,"      gosub __loadposition0__\r\n");
fprintf(f,"      goto __showboard__\r\n");

// eeprom-read, table-readtable
```

```
   static const int data_size =
sizeof(SETW_data)+sizeof(MOVEX_data)+sizeof(POINTS_data)+sizeof(OPNING_data);
   fprintf(f,"__read_static_data__:\r\n");
   fprintf(f,"      for b0 = 0 to 0x%02x\r\n",data_size-1);
   fprintf(f,"         read b0,b1\r\n");
   fprintf(f,"         w1 = b0+0x%02x\r\n",SETW);
   fprintf(f,"         put w1,b1\r\n");
   fprintf(f,"      next\r\n");
   fprintf(f,"      return\r\n");
   fprintf(f,"\r\n");
   for (int i=0;i<sizeof(SETW_data);i++)
   {
      fprintf(f,"      eeprom (0x%02x) ;SETW: 0x%02x\r\n",SETW_data[i],SETW+i);
   }
   for (int i=0;i<sizeof(MOVEX_data);i++)
   {
      fprintf(f,"      eeprom (0x%02x) ;MOVEX: 0x%02x\r\n",MOVEX_data[i],MOVEX+i);
   }
   for (int i=0;i<sizeof(POINTS_data);i++)
   {
      fprintf(f,"      eeprom (0x%02x) ;POINTS:
0x%02x\r\n",POINTS_data[i],POINTS+i);
   }
   for (int i=0;i<sizeof(OPNING_data);i++)
   {
      fprintf(f,"      eeprom (0x%02x) ;OPNING:
0x%02x\r\n",OPNING_data[i],OPNING+i);
   }
   fclose(f);
}

void chess( void )
{
                LDAi_   (0x00); // level 1
                STA     (LEVEL1);
                LDAi_   (0xFF); // level 1
                STA     (LEVEL2);
                LDAi_   (0x00);
                STA     (REV);
                LDXi_   (0x1F);             // clear board
                LDAi_   (0xCC);
INITCLEAR_      STAx    (BOARD,X);
                DEX;
                BPL     (INITCLEAR);

//
//
CHESS_BEGIN_    CLD;                        // INITIALIZE
                LDXi_   (0xFF);             // TWO STACKS
                TXS;
                LDXi_   (0xC8);
                STX     (SP2);
//
//       ROUTINES TO LIGHT LED
//       DISPLAY AND GET KEY
//       FROM KEYBOARD
//
                JSR_    (POUT);             // DISPLAY AND
```

```
                JSR_    (KIN);              // GET INPUT   *** my routine waits
for a keypress
//              CMP     (OLDKY);            // KEY IN ACC  *** no need to debounce
//              BEQ     (OUT);              // (DEBOUNCE)
//              STA     (OLDKY);
                ANDi_   (0x5F);             // convert to upper
                CMPi    (0x58);             // [X] level 1 (super blitz)
                BNE     (TESTLEVEL2);
                LDAi_   (0x00);             // level 1
                STA     (LEVEL1);
                LDAi_   (0xFF);             // level 1
                STA     (LEVEL2);
                LDAi_   (0x11);             // indicate level 1
                JMP     (CLDSP2);
TESTLEVEL2_     CMPi    (0x59);             // [Y] level 2 (blitz)
                BNE     (TESTLEVEL3);
                LDAi_   (0x00);             // level 2
                STA     (LEVEL1);
                LDAi_   (0xFB);             // level 2
                STA     (LEVEL2);
                LDAi_   (0x22);             // indicate level 2
                JMP     (CLDSP2);
TESTLEVEL3_     CMPi    (0x5A);             // [Z] level 3 (normal)
                BNE     (TESTSAVE);
                LDAi_   (0x08);             // level 3
                STA     (LEVEL1);
                LDAi_   (0xFB);             // level 3
                STA     (LEVEL2);
                LDAi_   (0x33);             // indicate level 3
                JMP     (CLDSP2);
TESTSAVE_       CMPi    (0x53);             // [S] save position
                BNE     (TESTLOAD);
                JSR_    ("__saveposition__");  // save the board and the reverse
flag
                LDAi_   (0x55);             // indicate saved
                JMP     (CLDSP2);
TESTLOAD_       CMPi    (0x4C);             // [L] load saved position
                BNE     (TESTRESTORE);
                JSR_    ("__loadposition__");  // load the board and the reverse
flag
                LDAi_   (0x88);             // indicate loaded
                JMP     (CLDSP2);
TESTRESTORE_    CMPi    (0x52);             // [R] load saved position
                BNE     (TESTUNDO);
                JSR_    ("__restoreposition__");  // load the board and the
reverse flag
                LDAi_   (0x88);             // indicate loaded
                JMP     (CLDSP2);
TESTUNDO_       CMPi    (0x55);             // [U] undo user move
                BNE     (SETUP);
                JSR_    ("__undoposition__");  // load the board and the reverse
flag
                LDAi_   (0x88);             // indicate loaded
                JMP     (CLDSP2);
//
SETUP_          ANDi_   (0x4F);             // MASK 0-7, AND ALPHA'S (moved from
KIN)
                CMPi    (0x43);             // [C]
                BNE     (NOSET);            // SET UP
```

```
                LDXi_   (0x1F);             // BOARD
WHSET_          LDAf    (SETW,X);           // FROM
                STAx    (BOARD,X);          // SETW
                DEX;
                BPL     (WHSET);
                LDXi_   (0x1B);             // *ADDED
                STX     (OMOVE);            // INITS TO 0xFF
                LDAi_   (0x00);             // added (igm)
                STA     (REV);              // computer plays white
                JSR_    ("__dosaveposition__"); // save for undo
                LDAi_   (0xCC);             // Display CCC
                JMP     (CLDSP);            // was BNE (igm)
//
NOSET_          CMPi    (0x45);             // [E]
                BNE     (NOREV);            // REVERSE
                JSR_    (REVERSE);          // BOARD IS
                SEC;
                LDAi_   (0x01);
                SBC     (REV);
                STA     (REV);              // TOGGLE REV FLAG
                JSR_    ("__dosaveposition__"); // save for undo
                LDAi_   (0xEE);             // IS
                JMP     (CLDSP);            // was BNE (igm)
//
NOREV_          CMPi    (0x40);             // [P] (P is 0x50 but masked with
0x4f)
                BNE     (NOGO);
                JSR     (GO);               // PLAY CHESS
                JSR_    ("__dosaveposition__"); // save for undo
CLDSP_          JSR_    ("__showboard__");  // display the whole board
CLDSP2_         STA     (DIS1);             // DISPLAY
                STA     (DIS2);             // ACROSS
                STA     (DIS3);             // DISPLAY
                JMP     (CHESS_BEGIN);
//
NOGO_           CMPi    (0x0D);             // [Enter]
                BNE     (NOMV);             // MOVE MAN
                JSR     (MOVE);             // AS ENTERED
                JSR_    ("__showboard__");  // display the whole board
                JMP     (DISP);             //
NOMV_           CMPi    (0x41);             // [Q] ***Added to allow game exit***
                BEQ     (DONE);             // quit the game, exit back to system.
                JMP     (INPUT);            //
DONE_           JMP     (RESTART_CHESS);    // clean start
//
//
JANUS_          LDX     (STATE);
                BMI     (NOCOUNT);
//
//       THIS ROUTINE COUNTS OCCURRENCES
//       IT DEPENDS UPON STATE TO INDEX
//       THE CORRECT COUNTERS
//
/*COUNTS_*/     LDA     (PIECE);
                BEQ     (OVER);             // IF STATE=8
                CPXi    (0x08);             // DO NOT COUNT
                BNE     (OVER);             // BLK MAX CAP
                CMP     (BMAXP);            // MOVES FOR
                BEQ     (XRT);              // WHITE
```

```
//
OVER_           INCx    (MOB,X);            // MOBILITY
                CMPi    (0x01);             //  + QUEEN
                BNE     (NOQ);              // FOR TWO
                INCx    (MOB,X);
//
NOQ_            BVC     (NOCAP);
                LDYi    (0x0F);             // CALCULATE
                LDA     (SQUARE);           // POINTS
ELOOP_          CMPx    (BK,Y);             // CAPTURED
                BEQ     (FOUN);             // BY THIS
                DEY;                        // MOVE
                BPL     (ELOOP);
FOUN_           LDAf    (POINTS,Y);
                CMPx    (MAXC,X);
                BCC     (LESS);             // SAVE IF
                STYx    (PCAP,X);           // BEST THIS
                STAx    (MAXC,X);           // STATE
//
LESS_           CLC;
                PHP;                        // ADD TO
                ADCx    (CC,X);             // CAPTURE
                STAx    (CC,X);             // COUNTS
                PLP;
//
NOCAP_          CPXi    (0x04);
                BEQ     (ON4);
                BMI     (TREE);             //(=00 ONLY)
XRT_            RTS;
//
//      GENERATE FURTHER MOVES FOR COUNT
//      AND ANALYSIS
//
ON4_            LDA     (XMAXC);            // SAVE ACTUAL
                STA     (WCAP0);            // CAPTURE
                LDAi    (0x00);             // STATE=0
                STA     (STATE);
                JSR     (MOVE);             // GENERATE
                JSR_    (REVERSE);          // IMMEDIATE
                JSR     (GNMZ);             // REPLY MOVES
                JSR_    (REVERSE);
                LDAi    (0x08);             // STATE=8
                STA     (STATE);            // GENERATE
                JSR     (GNM);              // CONTINUATION
                JSR     (UMOVE);            // MOVES
                JMP     (STRATGY);
NOCOUNT_        CPXi    (0xF9);
                BNE     (TREE);
//
//      DETERMINE IF THE KING CAN BE
//      TAKEN, USED BY CHKCHK
//
                LDA     (BK);               // IS KING
                CMP     (SQUARE);           // IN CHECK?
                BNE     (RETJ);             // SET INCHEK=0
                LDAi    (0x00);             // IF IT IS
                STA     (INCHEK);
RETJ_           RTS;
//
```

```
//      IF A PIECE HAS BEEN CAPTURED BY
//      A TRIAL MOVE, GENERATE REPLIES &
//      EVALUATE THE EXCHANGE GAIN/LOSS
//
TREE_           BVC     (RETJ);             // NO CAP
                LDYi    (0x07);             // (PIECES)
                LDA     (SQUARE);
LOOPX_          CMPx    (BK,Y);
                BEQ     (FOUNX);
                DEY;
                BEQ     (RETJ);             // (KING)
                BPL     (LOOPX);            // SAVE
FOUNX_          LDAf    (POINTS,Y);         // BEST CAP
                CMPx    (BCAP0,X);          // AT THIS
                BCC     (NOMAX);            // LEVEL
                STAx    (BCAP0,X);
NOMAX_          DEC     (STATE);
                LDA     (LEVEL2);           // IF STATE=FB  (WRF, was LDAi
(0xFB);)
                CMP     (STATE);            // TIME TO TURN
                BEQ     (UPTREE);           // AROUND
                JSR     (GENRM);            // GENERATE FURTHER
UPTREE_         INC     (STATE);            // CAPTURES
                RTS;
//
//      THE PLAYER'S MOVE IS INPUT
//
INPUT_          CMPi    (0x08);             // NOT A LEGAL
                BCS     (ERROR);            // SQUARE #
                JSR_    (DISMV);
                JMP     (DISP);             // fall through
ERROR_          JMP     (CHESS_BEGIN);
//
// display
//
DISP_           LDXi    (0x1F);
SEARCH_         LDAx    (BOARD,X);
                CMP     (DIS2);
                BEQ     (HERE);             // DISPLAY
                DEX;                        // PIECE AT
                BPL     (SEARCH);           // FROM
HERE_           STX     (DIS1);             // SQUARE
                STX     (PIECE);
                JMP     (CHESS_BEGIN);
//
//      GENERATE ALL MOVES FOR ONE
//      SIDE, CALL JANUS AFTER EACH
//      ONE FOR NEXT STEP
//
GNMZ_           LDXi    (0x10);             // CLEAR
//
GNMX_           LDAi    (0x00);             // COUNTERS
CLEAR_          STAx    (COUNT,X);
                DEX;
                BPL     (CLEAR);
//
GNM_            LDAi    (0x10);             // SET UP
                STA     (PIECE);            // PIECE
NEWP_           DEC     (PIECE);            // NEW PIECE
```

```
                BPL     (NEX);              // ALL DONE?
                RTS;                        //    -YES
//
NEX_            JSR_    (RESET);            // READY
                LDY     (PIECE);            // GET PIECE
                LDXi    (0x08);
                STX     (MOVEN);            // COMMON START
                CPYi    (0x08);             // WHAT IS IT?
                BPL     (PAWN);             // PAWN
                CPYi    (0x06);
                BPL     (KNIGHT);           // KNIGHT
                CPYi    (0x04);
                BPL     (BISHOP);           // BISHOP
                CPYi    (0x01);
                BEQ     (QUEEN);            // QUEEN
                BPL     (ROOK);             // ROOK
//
KING_           JSR     (SNGMV);            // MUST BE KING!
                BNE     (KING);             // MOVES
                BEQ     (NEWP);             // 8 TO 1
QUEEN_          JSR     (LINE);
                BNE     (QUEEN);            // MOVES
                BEQ     (NEWP);             // 8 TO 1
//
ROOK_           LDXi    (0x04);
                STX     (MOVEN);            // MOVES
AGNR_           JSR     (LINE);             // 4 TO 1
                BNE     (AGNR);
                BEQ     (NEWP);
//
BISHOP_         JSR     (LINE);
                LDA     (MOVEN);            // MOVES
                CMPi    (0x04);             // 8 TO 5
                BNE     (BISHOP);
                BEQ     (NEWP);
//
KNIGHT_         LDXi    (0x10);
                STX     (MOVEN);            // MOVES
AGNN_           JSR     (SNGMV);            // 16 TO 9
                LDA     (MOVEN);
                CMPi    (0x08);
                BNE     (AGNN);
                BEQ     (NEWP);
//
PAWN_           LDXi    (0x06);
                STX     (MOVEN);
P1_             JSR     (CMOVE);            // RIGHT CAP?
                BVC     (P2);
                BMI     (P2);
                JSR     (JANUS);            // YES
P2_             JSR_    (RESET);
                DEC     (MOVEN);            // LEFT CAP?
                LDA     (MOVEN);
                CMPi    (0x05);
                BEQ     (P1);
P3_             JSR     (CMOVE);            // AHEAD
                BVS     (NEWP);             // ILLEGAL
                BMI     (NEWP);
                JSR     (JANUS);
```

```
                LDA     (SQUARE);           // GETS TO
                ANDi_   (0xF0);             // 3RD RANK?
                CMPi    (0x20);
                BEQ     (P3);               // DO DOUBLE
                BRA     (NEWP);             // JMP (NEWP);
//
//      CALCULATE SINGLE STEP MOVES
//      FOR K,N
//
SNGMV_          JSR     (CMOVE);            // CALC MOVE
                BMI     (ILL1);             // -IF LEGAL
                JSR     (JANUS);            // -EVALUATE
ILL1_           JSR_    (RESET);
                DEC     (MOVEN);
                RTS;
//
//     CALCULATE ALL MOVES DOWN A
//     STRAIGHT LINE FOR Q,B,R
//
LINE_           JSR     (CMOVE);            // CALC MOVE
                BCC     (OVL);              // NO CHK
                BVC     (LINE);             // NOCAP
OVL_            BMI     (ILL);              // RETURN
                PHP;
                JSR     (JANUS);            // EVALUATE POSN
                PLP;
                BVC     (LINE);             // NOT A CAP
ILL_            JSR_    (RESET);            // LINE STOPPED
                DEC     (MOVEN);            // NEXT DIR
                RTS;
//
//      EXCHANGE SIDES FOR REPLY
//      ANALYSIS
//
REVERSE_        LDXi_   (0x0F);
ETC_            SEC;
                LDYx_   (BK,X);             // SUBTRACT
                LDAi_   (0x77);             // POSITION
                SBCx    (BOARD,X);          // FROM 77
                STAx    (BK,X);
                STYx    (BOARD,X);          // AND
                SEC;
                LDAi_   (0x77);             // EXCHANGE
                SBCx    (BOARD,X);          // PIECES
                STAx    (BOARD,X);
                DEX;
                BPL     (ETC);
                RTS_;
//
//        CMOVE CALCULATES THE TO SQUARE
//        USING SQUARE AND THE MOVE
//       TABLE  FLAGS SET AS FOLLOWS_
//       N - ILLEGAL MOVE
//       V - CAPTURE (LEGAL UNLESS IN CH)
//       C - ILLEGAL BECAUSE OF CHECK
//       [MY THANKS TO JIM BUTTERFIELD
//        WHO WROTE THIS MORE EFFICIENT
//        VERSION OF CMOVE]
//
```

```
CMOVE_          LDA     (SQUARE);           // GET SQUARE
                LDX     (MOVEN);            // MOVE POINTER
                CLC;
                ADCf    (MOVEX,X);          // MOVE LIST
                STA     (SQUARE);           // NEW POS'N
                ANDi    (0x88);
                BNE     (ILLEGAL);          // OFF BOARD
                LDA     (SQUARE);
                LDXi    (0x20);
LOOP_           DEX;                        // IS TO
                BMI     (NO);               // SQUARE
                CMPx    (BOARD,X);          // OCCUPIED?
                BNE     (LOOP);
                CPXi    (0x10);             // BY SELF?
                BMI     (ILLEGAL);
//              LDAi    (0x7F);             // MUST BE CAP!
//              ADCi    (0x01);             // SET V FLAG
                SEV;    LDAi(0x80);         // Avoid problematic V emulation
                JMP     (SPX);              // (JMP, was BVS [igm])
//
NO_             CLV;                        // NO CAPTURE
//
SPX_            LDA     (STATE);            // SHOULD WE
                BMI     (RETL);             // DO THE
                CMP     (LEVEL1);           // CHECK CHECK? (WRF_ was CMPi
(0x08);)
                BPL     (RETL);
//
//       CHKCHK REVERSES SIDES
//       AND LOOKS FOR A KING
//       CAPTURE TO INDICATE
//       ILLEGAL MOVE BECAUSE OF
//       CHECK  SINCE THIS IS
//       TIME CONSUMING, IT IS NOT
//       ALWAYS DONE
//
/*CHKCHK_*/     PHA;                        // STATE
                PHP;
                LDAi    (0xF9);
                STA     (STATE);            // GENERATE
                STA     (INCHEK);           // ALL REPLY
                JSR     (MOVE);             // MOVES TO
                JSR_    (REVERSE);          // SEE IF KING
                JSR     (GNM);              // IS IN
                JSR     (RUM);              // CHECK
                PLP;
                PLA;
                STA     (STATE);
                LDA     (INCHEK);
                BMI     (RETL);             // NO - SAFE
                SEC;                        // YES - IN CHK
                LDAi    (0xFF);
                RTS;
//
RETL_           CLC;                        // LEGAL
                LDAi    (0x00);             // RETURN
                RTS;
//
ILLEGAL_        LDAi    (0xFF);
```

```
                CLC;                        // ILLEGAL
                CLV;                        // RETURN
                RTS;
//
//       REPLACE PIECE ON CORRECT SQUARE
//
RESET_          LDX     (PIECE);            // GET LOGAT
                LDAx    (BOARD,X);          // FOR PIECE
                STA     (SQUARE);           // FROM BOARD
                RTS_;
//
//
GENRM_          JSR     (MOVE);             // MAKE MOVE
                JSR_    (REVERSE);          // REVERSE BOARD
                JSR     (GNM);              // GENERATE MOVES
RUM_            JSR_    (REVERSE);          // REVERSE BACK
//
//       ROUTINE TO UNMAKE A MOVE MADE BY
//         MOVE
//
UMOVE_          TSX;                        // UNMAKE MOVE
                STX     (SP1);
                LDX     (SP2);              // EXCHANGE
                TXS;                        // STACKS
                PLA;                        // MOVEN
                STA     (MOVEN);
                PLA;                        // CAPTURED
                STA     (PIECE);            // PIECE
                TAX;
                PLA;                        // FROM SQUARE
                STAx    (BOARD,X);
                PLA;                        // PIECE
                TAX;
                PLA;                        // TO SOUARE
                STA     (SQUARE);
                STAx    (BOARD,X);
                JMP     (STRV);
//
//       THIS ROUTINE MOVES PIECE
//       TO SQUARE, PARAMETERS
//       ARE SAVED IN A STACK TO UNMAKE
//       THE MOVE LATER
//
MOVE_           TSX;
                STX     (SP1);              // SWITCH
                LDX     (SP2);              // STACKS
                TXS;
                LDA     (SQUARE);
                PHA;                        // TO SQUARE
                TAY;
                LDXi    (0x1F);
CHECK_          CMPx    (BOARD,X);          // CHECK FOR
                BEQ     (TAKE);             // CAPTURE
                DEX;
                BPL     (CHECK);
TAKE_           LDAi    (0xCC);
                STAx    (BOARD,X);
                TXA;                        // CAPTURED
                PHA;                        // PIECE
```

```
                LDX     (PIECE);
                LDAx    (BOARD,X);
                STYx    (BOARD,X);          // FROM
                PHA;                        // SQUARE
                TXA;
                PHA;                        // PIECE
                LDA     (MOVEN);
                PHA;                        // MOVEN

//
//  Fortunately when we swap stacks we jump here and swap back before
//  returning. The original code does this so we can take advantage
//  on the picaxe and implement a stack and calling/return mechanism
//  that uses sratchpad to store an ID number for each return address.
//  This allows a subroutine call depth much greater than 8 (the limit
//  of the picaxe).
//
STRV_           TSX;
                STX     (SP2);              // SWITCH
                LDX     (SP1);              // STACKS
                TXS;                        // BACK
                RTS;
//
//       CONTINUATION OF SUB STRATGY
//       -CHECKS FOR CHECK OR CHECKMATE
//       AND ASSIGNS VALUE TO MOVE
//
CKMATE_         LDX     (BMAXC);            // CAN BLK CAP
                CPXf    (POINTS,"0");       // MY KING?
                BNE     (NOCHEK);
                LDAi    (0x00);             // GULP!
                JMP     (RETV);             // DUMB MOVE! was BEQ (igm)
//
NOCHEK_         LDX     (BMOB);             // IS BLACK
                BNE     (RETV);             // UNABLE TO
                LDX     (WMAXP);            // MOVE AND
                BNE     (RETV);             // KING IN CH?
                LDAi    (0xFF);             // YES! MATE
//
RETV_           LDXi    (0x04);             // RESTORE
                STX     (STATE);            // STATE=4
//
//       THE VALUE OF THE MOVE (IN ACCU)
//       IS COMPARED TO THE BEST MOVE AND
//       REPLACES IT IF IT IS BETTER
//
/*PUSH_*/       CMP     (BESTV);            // IS THIS BEST
                BCC     (RETP);             // MOVE SO FAR?
                BEQ     (RETP);
                STA     (BESTV);            // YES!
                LDA_    (PIECE);            // SAVE IT
                STA     (BESTP);
                LDA_    (SQUARE);
                STA     (BESTM);            // FLASH DISPLAY
RETP_           LDAi_   ('.');              // print ... instead of flashing disp
                fprintf(f,"      sertxd (reg_a)\r\n");
                RTS;
//
//       MAIN PROGRAM TO PLAY CHESS
```

```
//       PLAY FROM OPENING OR THINK
//
GO_             LDX     (OMOVE);            // OPENING?
                BMI     (NOOPEN);           // -NO   *ADD CHANGE FROM BPL
                LDA     (DIS3);             // -YES WAS
                CMPf    (OPNING,X);         // OPPONENT'S
                BNE     (END);              // MOVE OK?
                DEX;
                LDAf    (OPNING,X);         // GET NEXT
                STA     (DIS1);             // CANNED
                DEX;                        // OPENING MOVE
                LDAf    (OPNING,X);
                STA     (DIS3);             // DISPLAY IT
                DEX;
                STX     (OMOVE);            // MOVE IT
                BNE     (MV2);              // (JMP)
//
END_            LDAi    (0xFF);             // *ADD - STOP CANNED MOVES
                STA     (OMOVE);            // FLAG OPENING
NOOPEN_         LDXi    (0x0C);             // FINISHED
                STX     (STATE);            // STATE=C
                STX     (BESTV);            // CLEAR BESTV
                LDXi    (0x14);             // GENERATE P
                JSR     (GNMX);             // MOVES
//
                LDXi    (0x04);             // STATE=4
                STX     (STATE);            // GENERATE AND
                JSR     (GNMZ);             // TEST AVAILABLE
//                                             MOVES
//
                LDX     (BESTV);            // GET BEST MOVE
                CPXi    (0x0F);             // IF NONE
                BCC     (MATE);             // OH OH!
//
MV2_            LDX     (BESTP);            // MOVE
                LDAx    (BOARD,X);          // THE
                STA     (BESTV);            // BEST
                STX     (PIECE);            // MOVE
                LDA     (BESTM);
                STA     (SQUARE);           // AND DISPLAY
                JSR     (MOVE);             // IT
                JSR_    ("__dosaveposition__"); // save for undo
                JSR_    ("__showboard__");  // display the whole board
                JMP     (CHESS_BEGIN);
//
MATE_           LDAi    (0xFF);             // RESIGN
                RTS;                        // OR STALEMATE
//
//       SUBROUTINE TO ENTER THE
//       PLAYER'S MOVE
//
DISMV_          LDXi    (0x04);             // ROTATE
DROL_           ASL     (DIS3);             // KEY
                ROL     (DIS2);             // INTO
                DEX;                        // DISPLAY
                BNE     (DROL);             //
                ORA_    (DIS3);
                STA     (DIS3);
                STA     (SQUARE);
```

```
                RTS_;
//
//       THE FOLLOWING SUBROUTINE ASSIGNS
//       A VALUE TO THE MOVE UNDER
//       CONSIDERATION AND RETURNS IT IN
//       THE ACCUMULATOR
//
STRATGY_        CLC;
                LDAi_   (0x80);
                ADC     (WMOB);             // PARAMETERS
                ADC     (WMAXC);            // WITH WEIGHT
                ADC     (WCC);              // OF 0.25
                ADC     (WCAP1);
                ADC     (WCAP2);
                SEC;
                SBC     (PMAXC);
                SBC     (PCC);
                SBC     (BCAP0);
                SBC     (BCAP1);
                SBC     (BCAP2);
                SBC     (PMOB);
                SBC     (BMOB);
                BCS     (POS);              // UNDERFLOW
                LDAi_   (0x00);             // PREVENTION
POS_            LSR;
                CLC;                        // **************
                ADCi    (0x40);
                ADC     (WMAXC);            // PARAMETERS
                ADC     (WCC);              // WITH WEIGHT
                SEC;                        // OF 0.5
                SBC     (BMAXC);
                LSR;                        // **************
                CLC;
                ADCi    (0x90);
                ADC     (WCAP0);            // PARAMETERS
                ADC     (WCAP0);            // WITH WEIGHT
                ADC     (WCAP0);            // OF 1.0
                ADC     (WCAP0);
                ADC     (WCAP1);
                SEC;                        // [UNDER OR OVER-
                SBC     (BMAXC);            // FLOW MAY OCCUR
                SBC     (BMAXC);            // FROM THIS
                SBC     (BMCC);             // SECTION]
                SBC     (BMCC);
                SBC     (BCAP1);
                LDX     (SQUARE);           // ***************
                CPXi    (0x33);
                BEQ     (POSN);             // POSITION
                CPXi    (0x34);             // BONUS FOR
                BEQ     (POSN);             // MOVE TO
                CPXi    (0x22);             // CENTRE
                BEQ     (POSN);             // OR
                CPXi    (0x25);             // OUT OF
                BEQ     (POSN);             // BACK RANK
                LDX     (PIECE);
                BEQ     (NOPOSN);
                LDYx    (BOARD,X);
                CPYi    (0x10);
                BPL     (NOPOSN);
```

```
POSN_           CLC;
                ADCi    (0x02);
NOPOSN_         JMP     (CKMATE);           // CONTINUE
//
//
POUT_           LDA_   (DIS1);
                fprintf(f,"      gosub __hexbyte__\r\n");
                fprintf(f,"      sertxd (\" \")\r\n");
                LDA_   (DIS2);
                fprintf(f,"      gosub __hexbyte__\r\n");
                fprintf(f,"      sertxd (\" \")\r\n");
                LDA_   (DIS3);
                fprintf(f,"      gosub __hexbyte__\r\n");
                fprintf(f,"      sertxd (cr,lf)\r\n");
                RTS_;

KIN_            LDAi    ('?');
                fprintf(f,"      sertxd (reg_a)\r\n");
                fprintf(f,"      serrxd reg_a\r\n");
                RTS_;
//
//
RESTART_CHESS_  fprintf(f,"      reset\r\n");
}

int main(int argc, char *argv[])
{
   init();
   chess();
   done();
   return 0;
}
```

Appendix C – Generated PICAXE Source Code for 28X2

```
#rem
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************

  Kim-1 MicroChess (c) 1976-2005 Peter Jennings, www.benlo.com
  6502 emulation   (c) 2005 Bill Forster
  28X2 emulation   (c) 2015 Ian Mitchell

  Runs an emulation of the Kim-1 Microchess on the PICAXE 28X2
  microcontroller. Based on an idea from Bill Forster to emulate
  6502 microprocessor instructions in C. The program is created
  by running 28X2Microchess.exe. This file (28X2Microchess.bas)
  is generated and can be uploaded to a 28X2.

**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************

 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:
 1. Redistributions of source code must retain the above copyright
       notice, this list of conditions and the following disclaimer.
 2. Redistributions in binary form must reproduce the above copyright
       notice, this list of conditions and the following disclaimer in the
       documentation and/or other materials provided with the distribution.
 3. The name of the author may not be used to endorse or promote products
       derived from this software without specific prior written permission.

 THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS OR
 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
 IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES LOSS OF USE,
 DATA, OR PROFITS OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#endrem

#picaxe 28x2
setfreq em64
;setfreq m16
symbol twobytes = w0
symbol reg_a = b2
symbol reg_x = b3
symbol reg_y = b4
symbol reg_v = b5
symbol reg_f = b6
symbol reg_cy = b7
```

```
symbol reg_fc = w3
symbol temp = b8

symbol _row = b9
symbol _col = b10
symbol _loc = b11
symbol _pindex = b12
symbol _reverse = b13
symbol _p = b14

      gosub __read_static_data__
      reg_a = 0x00
      put 0x00,reg_a
      reg_a = 0xff
      put 0x01,reg_a
      reg_a = 0x00
      put 0xb7,reg_a
      reg_x = 0x1f
      reg_a = 0xcc
_INITCLEAR_:
      temp = reg_x+0x50 : put temp,reg_a
      dec reg_x : reg_f = reg_x
      if reg_f<0x80 then goto _INITCLEAR_
_CHESS_BEGIN_:
      reg_x = 0xff
      ptr = reg_x+0x100
      reg_x = 0xc8
      put 0xb2,reg_x
      gosub _POUT_
      gosub _KIN_
      reg_a = reg_a&0x5f
      reg_fc = reg_a+0xa8 ; CMPi(0x58)
      if reg_f!=0 then goto _TESTLEVEL2_
      reg_a = 0x00
      put 0x00,reg_a
      reg_a = 0xff
      put 0x01,reg_a
      reg_a = 0x11
      goto _CLDSP2_
_TESTLEVEL2_:
      reg_fc = reg_a+0xa7 ; CMPi(0x59)
      if reg_f!=0 then goto _TESTLEVEL3_
      reg_a = 0x00
      put 0x00,reg_a
      reg_a = 0xfb
      put 0x01,reg_a
      reg_a = 0x22
      goto _CLDSP2_
_TESTLEVEL3_:
      reg_fc = reg_a+0xa6 ; CMPi(0x5a)
      if reg_f!=0 then goto _TESTSAVE_
      reg_a = 0x08
      put 0x00,reg_a
      reg_a = 0xfb
      put 0x01,reg_a
      reg_a = 0x33
      goto _CLDSP2_
_TESTSAVE_:
      reg_fc = reg_a+0xad ; CMPi(0x53)
```

```
      if reg_f!=0 then goto _TESTLOAD_
      gosub __saveposition__
      reg_a = 0x55
      goto _CLDSP2_
_TESTLOAD_:
      reg_fc = reg_a+0xb4 ; CMPi(0x4c)
      if reg_f!=0 then goto _TESTRESTORE_
      gosub __loadposition__
      reg_a = 0x88
      goto _CLDSP2_
_TESTRESTORE_:
      reg_fc = reg_a+0xae ; CMPi(0x52)
      if reg_f!=0 then goto _TESTUNDO_
      gosub __restoreposition__
      reg_a = 0x88
      goto _CLDSP2_
_TESTUNDO_:
      reg_fc = reg_a+0xab ; CMPi(0x55)
      if reg_f!=0 then goto _SETUP_
      gosub __undoposition__
      reg_a = 0x88
      goto _CLDSP2_
_SETUP_:
      reg_a = reg_a&0x4f
      reg_fc = reg_a+0xbd ; CMPi(0x43)
      if reg_f!=0 then goto _NOSET_
      reg_x = 0x1f
_WHSET_:
      twobytes = reg_x+0x200 : get twobytes,reg_a : reg_f = reg_a
      temp = reg_x+0x50 : put temp,reg_a
      dec reg_x : reg_f = reg_x
      if reg_f<0x80 then goto _WHSET_
      reg_x = 0x1b
      put 0xdc,reg_x
      reg_a = 0x00
      put 0xb7,reg_a
      gosub __dosaveposition__
      reg_a = 0xcc
      goto _CLDSP_
_NOSET_:
      reg_fc = reg_a+0xbb ; CMPi(0x45)
      if reg_f!=0 then goto _NOREV_
      gosub _REVERSE_
      reg_cy = 1
      reg_a = 0x01
      get 0xb7,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      put 0xb7,reg_a
      gosub __dosaveposition__
      reg_a = 0xee
      goto _CLDSP_
_NOREV_:
      reg_fc = reg_a+0xc0 ; CMPi(0x40)
      if reg_f!=0 then goto _NOGO_
      @ptrdec = 0 : goto _GO_
_00:  gosub __dosaveposition__
_CLDSP_:
      gosub __showboard__
_CLDSP2_:
```

```
      put 0xfb,reg_a
      put 0xfa,reg_a
      put 0xf9,reg_a
      goto _CHESS_BEGIN_
_NOGO_:
      reg_fc = reg_a+0xf3 ; CMPi(0x0d)
      if reg_f!=0 then goto _NOMV_
      @ptrdec = 1 : goto _MOVE_
_01:  gosub __showboard__
      goto _DISP_
_NOMV_:
      reg_fc = reg_a+0xbf ; CMPi(0x41)
      if reg_f=0 then goto _DONE_
      goto _INPUT_
_DONE_:
      goto _RESTART_CHESS_
_JANUS_:
      get 0xb5,reg_x : reg_f = reg_x
      if reg_f>=0x80 then goto _NOCOUNT_
      get 0xb0,reg_a : reg_f = reg_a
      if reg_f=0 then goto _OVER_
      reg_fc = reg_x+0xf8 ; CPXi(0x08)
      if reg_f!=0 then goto _OVER_
      get 0xe6,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_f=0 then goto _XRT_
_OVER_:
      temp = reg_x+0xe3 : get temp,reg_f : inc reg_f : put temp,reg_f
      reg_fc = reg_a+0xff ; CMPi(0x01)
      if reg_f!=0 then goto _NOQ_
      temp = reg_x+0xe3 : get temp,reg_f : inc reg_f : put temp,reg_f
_NOQ_:
      if reg_v=0 then goto _NOCAP_
      reg_y = 0x0f : reg_f = reg_y
      get 0xb1,reg_a : reg_f = reg_a
_ELOOP_:
      reg_f = reg_y+0x60 : get reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1
      if reg_f=0 then goto _FOUN_
      dec reg_y : reg_f = reg_y
      if reg_f<0x80 then goto _ELOOP_
_FOUN_:
      twobytes = reg_y+0x231 : get twobytes,reg_a : reg_f = reg_a
      reg_f = reg_x+0xe4 : get reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1
      if reg_cy=0 then goto _LESS_
      temp = reg_x+0xe6 : put temp,reg_y
      temp = reg_x+0xe4 : put temp,reg_a
_LESS_:
      reg_cy = 0
      temp = reg_v<<1|reg_cy : temp = reg_f>>7<<3|temp : temp = reg_f max
1<<2|temp : @ptrdec = temp
      reg_f = reg_x+0xe5 : get reg_f,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      temp = reg_x+0xe5 : put temp,reg_a
      inc ptr : temp = @ptr : reg_f = temp<<4&0xc0: reg_cy = temp&0x01 : reg_v =
temp>>1&0x01
_NOCAP_:
      reg_fc = reg_x+0xfc ; CPXi(0x04)
      if reg_f=0 then goto _ON4_
```

```
      if reg_f>=0x80 then goto _TREE_
_XRT_:
      goto __return__
_ON4_:
      get 0xe8,reg_a : reg_f = reg_a
      put 0xdd,reg_a
      reg_a = 0x00 : reg_f = reg_a
      put 0xb5,reg_a
      @ptrdec = 2 : goto _MOVE_
_02:  gosub _REVERSE_
      @ptrdec = 3 : goto _GNMZ_
_03:  gosub _REVERSE_
      reg_a = 0x08 : reg_f = reg_a
      put 0xb5,reg_a
      @ptrdec = 4 : goto _GNM_
_04:  @ptrdec = 5 : goto _UMOVE_
_05:  goto _STRATGY_
_NOCOUNT_:
      reg_fc = reg_x+0x07 ; CPXi(0xf9)
      if reg_f!=0 then goto _TREE_
      get 0x60,reg_a : reg_f = reg_a
      get 0xb1,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_f!=0 then goto _RETJ_
      reg_a = 0x00 : reg_f = reg_a
      put 0xb4,reg_a
_RETJ_:
      goto __return__
_TREE_:
      if reg_v=0 then goto _RETJ_
      reg_y = 0x07 : reg_f = reg_y
      get 0xb1,reg_a : reg_f = reg_a
_LOOPX_:
      reg_f = reg_y+0x60 : get reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1
      if reg_f=0 then goto _FOUNX_
      dec reg_y : reg_f = reg_y
      if reg_f=0 then goto _RETJ_
      if reg_f<0x80 then goto _LOOPX_
_FOUNX_:
      twobytes = reg_y+0x231 : get twobytes,reg_a : reg_f = reg_a
      reg_f = reg_x+0xe2 : get reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1
      if reg_cy=0 then goto _NOMAX_
      temp = reg_x+0xe2 : put temp,reg_a
_NOMAX_:
      get 0xb5,reg_f : dec reg_f : put 0xb5,reg_f
      get 0x01,reg_a : reg_f = reg_a
      get 0xb5,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_f=0 then goto _UPTREE_
      @ptrdec = 6 : goto _GENRM_
_UPTREE_:
_06:  get 0xb5,reg_f : inc reg_f : put 0xb5,reg_f
      goto __return__
_INPUT_:
      reg_fc = reg_a+0xf8 ; CMPi(0x08)
      if reg_cy!=0 then goto _ERROR_
      gosub _DISMV_
      goto _DISP_
_ERROR_:
```

```
      goto _CHESS_BEGIN_
_DISP_:
      reg_x = 0x1f : reg_f = reg_x
_SEARCH_:
      reg_f = reg_x+0x50 : get reg_f,reg_a : reg_f = reg_a
      get 0xfa,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_f=0 then goto _HERE_
      dec reg_x : reg_f = reg_x
      if reg_f<0x80 then goto _SEARCH_
_HERE_:
      put 0xfb,reg_x
      put 0xb0,reg_x
      goto _CHESS_BEGIN_
_GNMZ_:
      reg_x = 0x10 : reg_f = reg_x
_GNMX_:
      reg_a = 0x00 : reg_f = reg_a
_CLEAR_:
      temp = reg_x+0xde : put temp,reg_a
      dec reg_x : reg_f = reg_x
      if reg_f<0x80 then goto _CLEAR_
_GNM_:
      reg_a = 0x10 : reg_f = reg_a
      put 0xb0,reg_a
_NEWP_:
      get 0xb0,reg_f : dec reg_f : put 0xb0,reg_f
      if reg_f<0x80 then goto _NEX_
      goto __return__
_NEX_:
      gosub _RESET_
      get 0xb0,reg_y : reg_f = reg_y
      reg_x = 0x08 : reg_f = reg_x
      put 0xb6,reg_x
      reg_fc = reg_y+0xf8 ; CPYi(0x08)
      if reg_f<0x80 then goto _PAWN_
      reg_fc = reg_y+0xfa ; CPYi(0x06)
      if reg_f<0x80 then goto _KNIGHT_
      reg_fc = reg_y+0xfc ; CPYi(0x04)
      if reg_f<0x80 then goto _BISHOP_
      reg_fc = reg_y+0xff ; CPYi(0x01)
      if reg_f=0 then goto _QUEEN_
      if reg_f<0x80 then goto _ROOK_
_KING_:
      @ptrdec = 7 : goto _SNGMV_
_07:  if reg_f!=0 then goto _KING_
      if reg_f=0 then goto _NEWP_
_QUEEN_:
      @ptrdec = 8 : goto _LINE_
_08:  if reg_f!=0 then goto _QUEEN_
      if reg_f=0 then goto _NEWP_
_ROOK_:
      reg_x = 0x04 : reg_f = reg_x
      put 0xb6,reg_x
_AGNR_:
      @ptrdec = 9 : goto _LINE_
_09:  if reg_f!=0 then goto _AGNR_
      if reg_f=0 then goto _NEWP_
_BISHOP_:
      @ptrdec = 10 : goto _LINE_
```

```
_10:  get 0xb6,reg_a : reg_f = reg_a
      reg_fc = reg_a+0xfc ; CMPi(0x04)
      if reg_f!=0 then goto _BISHOP_
      if reg_f=0 then goto _NEWP_
_KNIGHT_:
      reg_x = 0x10 : reg_f = reg_x
      put 0xb6,reg_x
_AGNN_:
      @ptrdec = 11 : goto _SNGMV_
_11:  get 0xb6,reg_a : reg_f = reg_a
      reg_fc = reg_a+0xf8 ; CMPi(0x08)
      if reg_f!=0 then goto _AGNN_
      if reg_f=0 then goto _NEWP_
_PAWN_:
      reg_x = 0x06 : reg_f = reg_x
      put 0xb6,reg_x
_P1_:
      @ptrdec = 12 : goto _CMOVE_
_12:  if reg_v=0 then goto _P2_
      if reg_f>=0x80 then goto _P2_
      @ptrdec = 13 : goto _JANUS_
_P2_:
_13:  gosub _RESET_
      get 0xb6,reg_f : dec reg_f : put 0xb6,reg_f
      get 0xb6,reg_a : reg_f = reg_a
      reg_fc = reg_a+0xfb ; CMPi(0x05)
      if reg_f=0 then goto _P1_
_P3_:
      @ptrdec = 14 : goto _CMOVE_
_14:  if reg_v!=0 then goto _NEWP_
      if reg_f>=0x80 then goto _NEWP_
      @ptrdec = 15 : goto _JANUS_
_15:  get 0xb1,reg_a : reg_f = reg_a
      reg_a = reg_a&0xf0
      reg_fc = reg_a+0xe0 ; CMPi(0x20)
      if reg_f=0 then goto _P3_
      goto _NEWP_
_SNGMV_:
      @ptrdec = 16 : goto _CMOVE_
_16:  if reg_f>=0x80 then goto _ILL1_
      @ptrdec = 17 : goto _JANUS_
_ILL1_:
_17:  gosub _RESET_
      get 0xb6,reg_f : dec reg_f : put 0xb6,reg_f
      goto __return__
_LINE_:
      @ptrdec = 18 : goto _CMOVE_
_18:  if reg_cy=0 then goto _OVL_
      if reg_v=0 then goto _LINE_
_OVL_:
      if reg_f>=0x80 then goto _ILL_
      temp = reg_v<<1|reg_cy : temp = reg_f>>7<<3|temp : temp = reg_f max
1<<2|temp : @ptrdec = temp
      @ptrdec = 19 : goto _JANUS_
_19:  inc ptr : temp = @ptr : reg_f = temp<<4&0xc0: reg_cy = temp&0x01 : reg_v =
temp>>1&0x01
      if reg_v=0 then goto _LINE_
_ILL_:
      gosub _RESET_
```

```
      get 0xb6,reg_f : dec reg_f : put 0xb6,reg_f
      goto __return__
_REVERSE_:
      reg_x = 0x0f
_ETC_:
      reg_cy = 1
      reg_f = reg_x+0x60 : get reg_f,reg_y
      reg_a = 0x77
      reg_f = reg_x+0x50 : get reg_f,reg_f : reg_f = not reg_f: reg_fc =
reg_a+reg_f+reg_cy : reg_a = reg_f
      temp = reg_x+0x60 : put temp,reg_a
      temp = reg_x+0x50 : put temp,reg_y
      reg_cy = 1
      reg_a = 0x77
      reg_f = reg_x+0x50 : get reg_f,reg_f : reg_f = not reg_f: reg_fc =
reg_a+reg_f+reg_cy : reg_a = reg_f
      temp = reg_x+0x50 : put temp,reg_a
      dec reg_x : reg_f = reg_x
      if reg_f<0x80 then goto _ETC_
      return
_CMOVE_:
      get 0xb1,reg_a : reg_f = reg_a
      get 0xb6,reg_x : reg_f = reg_x
      reg_cy = 0
      twobytes = reg_x+0x220 : get twobytes,reg_f : reg_fc = reg_a+reg_f+reg_cy :
reg_a = reg_f
      put 0xb1,reg_a
      reg_a = reg_a&0x88 : reg_f = reg_a
      if reg_f!=0 then goto _ILLEGAL_
      get 0xb1,reg_a : reg_f = reg_a
      reg_x = 0x20 : reg_f = reg_x
_LOOP_:
      dec reg_x : reg_f = reg_x
      if reg_f>=0x80 then goto _NO_
      reg_f = reg_x+0x50 : get reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1
      if reg_f!=0 then goto _LOOP_
      reg_fc = reg_x+0xf0 ; CPXi(0x10)
      if reg_f>=0x80 then goto _ILLEGAL_
      reg_v = 1
      reg_a = 0x80 : reg_f = reg_a
      goto _SPX_
_NO_:
      reg_v = 0
_SPX_:
      get 0xb5,reg_a : reg_f = reg_a
      if reg_f>=0x80 then goto _RETL_
      get 0x00,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_f<0x80 then goto _RETL_
      @ptrdec = reg_a
      temp = reg_v<<1|reg_cy : temp = reg_f>>7<<3|temp : temp = reg_f max
1<<2|temp : @ptrdec = temp
      reg_a = 0xf9 : reg_f = reg_a
      put 0xb5,reg_a
      put 0xb4,reg_a
      @ptrdec = 20 : goto _MOVE_
_20:  gosub _REVERSE_
      @ptrdec = 21 : goto _GNM_
_21:  @ptrdec = 22 : goto _RUM_
```

```
_22:  inc ptr : temp = @ptr : reg_f = temp<<4&0xc0: reg_cy = temp&0x01 : reg_v =
temp>>1&0x01
      inc ptr : reg_a = @ptr
      put 0xb5,reg_a
      get 0xb4,reg_a : reg_f = reg_a
      if reg_f>=0x80 then goto _RETL_
      reg_cy = 1
      reg_a = 0xff : reg_f = reg_a
      goto __return__
_RETL_:
      reg_cy = 0
      reg_a = 0x00 : reg_f = reg_a
      goto __return__
_ILLEGAL_:
      reg_a = 0xff : reg_f = reg_a
      reg_cy = 0
      reg_v = 0
      goto __return__
_RESET_:
      get 0xb0,reg_x : reg_f = reg_x
      reg_f = reg_x+0x50 : get reg_f,reg_a : reg_f = reg_a
      put 0xb1,reg_a
      return
_GENRM_:
      @ptrdec = 23 : goto _MOVE_
_23:  gosub _REVERSE_
      @ptrdec = 24 : goto _GNM_
_RUM_:
_24:  gosub _REVERSE_
_UMOVE_:
      reg_x = ptr-0x100
      put 0xb3,reg_x
      get 0xb2,reg_x : reg_f = reg_x
      ptr = reg_x+0x100
      inc ptr : reg_a = @ptr
      put 0xb6,reg_a
      inc ptr : reg_a = @ptr
      put 0xb0,reg_a
      reg_x = reg_a : reg_f = reg_a
      inc ptr : reg_a = @ptr
      temp = reg_x+0x50 : put temp,reg_a
      inc ptr : reg_a = @ptr
      reg_x = reg_a : reg_f = reg_a
      inc ptr : reg_a = @ptr
      put 0xb1,reg_a
      temp = reg_x+0x50 : put temp,reg_a
      goto _STRV_
_MOVE_:
      reg_x = ptr-0x100
      put 0xb3,reg_x
      get 0xb2,reg_x : reg_f = reg_x
      ptr = reg_x+0x100
      get 0xb1,reg_a : reg_f = reg_a
      @ptrdec = reg_a
      reg_y = reg_a : reg_f = reg_a
      reg_x = 0x1f : reg_f = reg_x
_CHECK_:
      reg_f = reg_x+0x50 : get reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1
```

```
      if reg_f=0 then goto _TAKE_
      dec reg_x : reg_f = reg_x
      if reg_f<0x80 then goto _CHECK_
_TAKE_:
      reg_a = 0xcc : reg_f = reg_a
      temp = reg_x+0x50 : put temp,reg_a
      reg_a = reg_x : reg_f = reg_x
      @ptrdec = reg_a
      get 0xb0,reg_x : reg_f = reg_x
      reg_f = reg_x+0x50 : get reg_f,reg_a : reg_f = reg_a
      temp = reg_x+0x50 : put temp,reg_y
      @ptrdec = reg_a
      reg_a = reg_x : reg_f = reg_x
      @ptrdec = reg_a
      get 0xb6,reg_a : reg_f = reg_a
      @ptrdec = reg_a
_STRV_:
      reg_x = ptr-0x100
      put 0xb2,reg_x
      get 0xb3,reg_x : reg_f = reg_x
      ptr = reg_x+0x100
      goto __return__
_CKMATE_:
      get 0xe4,reg_x : reg_f = reg_x
      twobytes = 0+0x231 : get twobytes,reg_f : reg_f = not reg_f : reg_fc =
reg_x+reg_f+1
      if reg_f!=0 then goto _NOCHEK_
      reg_a = 0x00 : reg_f = reg_a
      goto _RETV_
_NOCHEK_:
      get 0xe3,reg_x : reg_f = reg_x
      if reg_f!=0 then goto _RETV_
      get 0xee,reg_x : reg_f = reg_x
      if reg_f!=0 then goto _RETV_
      reg_a = 0xff : reg_f = reg_a
_RETV_:
      reg_x = 0x04 : reg_f = reg_x
      put 0xb5,reg_x
      get 0xfa,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_cy=0 then goto _RETP_
      if reg_f=0 then goto _RETP_
      put 0xfa,reg_a
      get 0xb0,reg_a
      put 0xfb,reg_a
      get 0xb1,reg_a
      put 0xf9,reg_a
_RETP_:
      reg_a = 0x2e
      sertxd (reg_a)
      goto __return__
_GO_:
      get 0xdc,reg_x : reg_f = reg_x
      if reg_f>=0x80 then goto _NOOPEN_
      get 0xf9,reg_a : reg_f = reg_a
      twobytes = reg_x+0x241 : get twobytes,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1
      if reg_f!=0 then goto _END_
      dec reg_x : reg_f = reg_x
      twobytes = reg_x+0x241 : get twobytes,reg_a : reg_f = reg_a
```

```
      put 0xfb,reg_a
      dec reg_x : reg_f = reg_x
      twobytes = reg_x+0x241 : get twobytes,reg_a : reg_f = reg_a
      put 0xf9,reg_a
      dec reg_x : reg_f = reg_x
      put 0xdc,reg_x
      if reg_f!=0 then goto _MV2_
_END_:
      reg_a = 0xff : reg_f = reg_a
      put 0xdc,reg_a
_NOOPEN_:
      reg_x = 0x0c : reg_f = reg_x
      put 0xb5,reg_x
      put 0xfa,reg_x
      reg_x = 0x14 : reg_f = reg_x
      @ptrdec = 25 : goto _GNMX_
_25:  reg_x = 0x04 : reg_f = reg_x
      put 0xb5,reg_x
      @ptrdec = 26 : goto _GNMZ_
_26:  get 0xfa,reg_x : reg_f = reg_x
      reg_fc = reg_x+0xf1 ; CPXi(0x0f)
      if reg_cy=0 then goto _MATE_
_MV2_:
      get 0xfb,reg_x : reg_f = reg_x
      reg_f = reg_x+0x50 : get reg_f,reg_a : reg_f = reg_a
      put 0xfa,reg_a
      put 0xb0,reg_x
      get 0xf9,reg_a : reg_f = reg_a
      put 0xb1,reg_a
      @ptrdec = 27 : goto _MOVE_
_27:  gosub __dosaveposition__
      gosub __showboard__
      goto _CHESS_BEGIN_
_MATE_:
      reg_a = 0xff : reg_f = reg_a
      goto __return__
_DISMV_:
      reg_x = 0x04 : reg_f = reg_x
_DROL_:
      get 0xf9,reg_f : reg_cy = reg_f>>7 : reg_f = reg_f<<1 : put 0xf9,reg_f
      get 0xfa,reg_f : temp = reg_f>>7 : reg_f = reg_f<<1|reg_cy : put 0xfa,reg_f
: reg_cy = temp
      dec reg_x : reg_f = reg_x
      if reg_f!=0 then goto _DROL_
      get 0xf9,reg_f : reg_a = reg_a|reg_f
      put 0xf9,reg_a
      put 0xb1,reg_a
      return
_STRATGY_:
      reg_cy = 0
      reg_a = 0x80
      get 0xeb,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0xec,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0xed,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0xe1,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0xdf,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      reg_cy = 1
      get 0xf0,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
```

```
      get 0xf1,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0xe2,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0xe0,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0xde,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0xef,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0xe3,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      if reg_cy!=0 then goto _POS_
      reg_a = 0x00
_POS_:
      reg_cy = reg_a&0x01 : reg_a = reg_a>>1 : reg_f = reg_a
      reg_cy = 0
      reg_fc = reg_a+0x40+reg_cy : reg_a = reg_f
      get 0xec,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0xed,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      reg_cy = 1
      get 0xe4,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      reg_cy = reg_a&0x01 : reg_a = reg_a>>1 : reg_f = reg_a
      reg_cy = 0
      reg_fc = reg_a+0x90+reg_cy : reg_a = reg_f
      get 0xdd,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0xdd,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0xdd,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0xdd,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0xe1,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      reg_cy = 1
      get 0xe4,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0xe4,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0xe5,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0xe5,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0xe0,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0xb1,reg_x : reg_f = reg_x
      reg_fc = reg_x+0xcd ; CPXi(0x33)
      if reg_f=0 then goto _POSN_
      reg_fc = reg_x+0xcc ; CPXi(0x34)
      if reg_f=0 then goto _POSN_
      reg_fc = reg_x+0xde ; CPXi(0x22)
      if reg_f=0 then goto _POSN_
      reg_fc = reg_x+0xdb ; CPXi(0x25)
      if reg_f=0 then goto _POSN_
      get 0xb0,reg_x : reg_f = reg_x
      if reg_f=0 then goto _NOPOSN_
      reg_f = reg_x+0x50 : get reg_f,reg_y : reg_f = reg_y
      reg_fc = reg_y+0xf0 ; CPYi(0x10)
      if reg_f<0x80 then goto _NOPOSN_
_POSN_:
      reg_cy = 0
      reg_fc = reg_a+0x02+reg_cy : reg_a = reg_f
```

```
_NOPOSN_:
      goto _CKMATE_
_POUT_:
      get 0xfb,reg_a
      gosub __hexbyte__
      sertxd (" ")
      get 0xfa,reg_a
      gosub __hexbyte__
      sertxd (" ")
      get 0xf9,reg_a
      gosub __hexbyte__
      sertxd (cr,lf)
      return
_KIN_:
      reg_a = 0x3f : reg_f = reg_a
      sertxd (reg_a)
      serrxd reg_a
      return
_RESTART_CHESS_:
      reset
__hexbyte__:
      temp = reg_a>>4+"0"
      gosub __nybble__
      temp = reg_a&0x0f+"0"
__nybble__:
      if temp>"9" then : temp = temp+7 : endif
      sertxd (temp)
      return
__showboard__:
      gosub __backupposition__
      sertxd (cr,lf)
      gosub __rownum__
      gosub __line__
      get 0xb7,_reverse
      for _row=0 to 7
         sertxd (#_row,"0|")
         for _col = 0 to 7
            _loc = _row<<4+_col
            for _pindex = 0 to 0x1f
               _p = _pindex+0x50
               get _p,_p
               if _p=_loc then
                  _p = _pindex>>4^_reverse
                  if _p=0 then sertxd ("W") : else : sertxd ("B") : endif
                  _p = _pindex&0x0f
                  lookup _p,("KQRRBBNNPPPPPPPP"),_p
                  sertxd (_p)
                  goto __next_location__
               endif
            next
            ; not found
            _p = _row^_col&1
            if _p=1 then : sertxd ("**") : else : sertxd ("  ") : endif
            __next_location__:
            sertxd ("|")
         next
         sertxd (#_row,"0",cr,lf)
         gosub __line__
      next
```

```
      gosub __rownum__
      return
__line__:
      sertxd ("  ") : for _p=1 to 25 : sertxd ("-") : next : sertxd (cr,lf)
      return
__rownum__:
      sertxd ("  ") : for _p=0 to 7 : sertxd (" 0",#_p) : next : sertxd (cr,lf)
      return
__return__:
      inc ptr
      branch
@ptr,(_00,_01,_02,_03,_04,_05,_06,_07,_08,_09,_10,_11,_12,_13,_14,_15,_16,_17,_18,
_19,_20,_21,_22,_23,_24,_25,_26,_27)
__saveposition__:
      write 0xff,0xff
      _loc = 0x64
__saveposition0__:
      ;_loc has eeprom address
      for temp = 0 to 0x1f
         _p = temp+0x50
         get _p,_p
         _pindex =_loc+temp
         write _pindex,_p
      next
      get 0xb7,_p
      inc _pindex
      write _pindex,_p
      get 0x00,_p
      inc _pindex
      write _pindex,_p
      get 0x01,_p
      inc _pindex
      write _pindex,_p
      return
__loadposition0__:
      ;_loc has eeprom address
      for temp = 0 to 0x1f
         _pindex =_loc+temp
         read _pindex,_p
         _pindex = temp+0x50
         put _pindex,_p
      next
      _pindex = _loc+0x20
      read _pindex,_p
      put 0xb7,_p
      inc _pindex
      read _pindex,_p
      put 0x00,_p
      inc _pindex
      read _pindex,_p
      put 0x01,_p
      return
__loadposition__:
      read 0xff,temp
      if temp!=0xff then
         sertxd (cr,lf,"Save first.",cr,lf)
         return
      endif
      _loc = 0x64
```

```
      gosub __loadposition0__
      goto __showboard__
__backupposition__:
      write 0xfe,0xff
      _loc = 0x8c
      goto __saveposition0__
__restoreposition__:
      read 0xfe,temp
      if temp!=0xff then
         sertxd (cr,lf,"Can't restore.",cr,lf)
         return
      endif
      _loc = 0x8c
      gosub __loadposition0__
      goto __showboard__
__dosaveposition__:
      write 0xfd,0xff
      _loc = 0xb4
      goto __saveposition0__
__undoposition__:
      read 0xfd,temp
      if temp!=0xff then
         sertxd (cr,lf,"Can't undo.",cr,lf)
         return
      endif
      _loc = 0xb4
      gosub __loadposition0__
      goto __showboard__
__read_static_data__:
      for b0 = 0 to 0x5c
         read b0,b1
         w1 = b0+0x200
         put w1,b1
      next
      return

      eeprom (0x03) ;SETW: 0x200
      eeprom (0x04) ;SETW: 0x201
      eeprom (0x00) ;SETW: 0x202
      eeprom (0x07) ;SETW: 0x203
      eeprom (0x02) ;SETW: 0x204
      eeprom (0x05) ;SETW: 0x205
      eeprom (0x01) ;SETW: 0x206
      eeprom (0x06) ;SETW: 0x207
      eeprom (0x10) ;SETW: 0x208
      eeprom (0x17) ;SETW: 0x209
      eeprom (0x11) ;SETW: 0x20a
      eeprom (0x16) ;SETW: 0x20b
      eeprom (0x12) ;SETW: 0x20c
      eeprom (0x15) ;SETW: 0x20d
      eeprom (0x14) ;SETW: 0x20e
      eeprom (0x13) ;SETW: 0x20f
      eeprom (0x73) ;SETW: 0x210
      eeprom (0x74) ;SETW: 0x211
      eeprom (0x70) ;SETW: 0x212
      eeprom (0x77) ;SETW: 0x213
      eeprom (0x72) ;SETW: 0x214
      eeprom (0x75) ;SETW: 0x215
      eeprom (0x71) ;SETW: 0x216
```

```
eeprom (0x76) ;SETW: 0x217
eeprom (0x60) ;SETW: 0x218
eeprom (0x67) ;SETW: 0x219
eeprom (0x61) ;SETW: 0x21a
eeprom (0x66) ;SETW: 0x21b
eeprom (0x62) ;SETW: 0x21c
eeprom (0x65) ;SETW: 0x21d
eeprom (0x64) ;SETW: 0x21e
eeprom (0x63) ;SETW: 0x21f
eeprom (0x00) ;MOVEX: 0x220
eeprom (0xf0) ;MOVEX: 0x221
eeprom (0xff) ;MOVEX: 0x222
eeprom (0x01) ;MOVEX: 0x223
eeprom (0x10) ;MOVEX: 0x224
eeprom (0x11) ;MOVEX: 0x225
eeprom (0x0f) ;MOVEX: 0x226
eeprom (0xef) ;MOVEX: 0x227
eeprom (0xf1) ;MOVEX: 0x228
eeprom (0xdf) ;MOVEX: 0x229
eeprom (0xe1) ;MOVEX: 0x22a
eeprom (0xee) ;MOVEX: 0x22b
eeprom (0xf2) ;MOVEX: 0x22c
eeprom (0x12) ;MOVEX: 0x22d
eeprom (0x0e) ;MOVEX: 0x22e
eeprom (0x1f) ;MOVEX: 0x22f
eeprom (0x21) ;MOVEX: 0x230
eeprom (0x0b) ;POINTS: 0x231
eeprom (0x0a) ;POINTS: 0x232
eeprom (0x06) ;POINTS: 0x233
eeprom (0x06) ;POINTS: 0x234
eeprom (0x04) ;POINTS: 0x235
eeprom (0x04) ;POINTS: 0x236
eeprom (0x04) ;POINTS: 0x237
eeprom (0x04) ;POINTS: 0x238
eeprom (0x02) ;POINTS: 0x239
eeprom (0x02) ;POINTS: 0x23a
eeprom (0x02) ;POINTS: 0x23b
eeprom (0x02) ;POINTS: 0x23c
eeprom (0x02) ;POINTS: 0x23d
eeprom (0x02) ;POINTS: 0x23e
eeprom (0x02) ;POINTS: 0x23f
eeprom (0x02) ;POINTS: 0x240
eeprom (0x99) ;OPNING: 0x241
eeprom (0x25) ;OPNING: 0x242
eeprom (0x0b) ;OPNING: 0x243
eeprom (0x25) ;OPNING: 0x244
eeprom (0x01) ;OPNING: 0x245
eeprom (0x00) ;OPNING: 0x246
eeprom (0x33) ;OPNING: 0x247
eeprom (0x25) ;OPNING: 0x248
eeprom (0x07) ;OPNING: 0x249
eeprom (0x36) ;OPNING: 0x24a
eeprom (0x34) ;OPNING: 0x24b
eeprom (0x0d) ;OPNING: 0x24c
eeprom (0x34) ;OPNING: 0x24d
eeprom (0x34) ;OPNING: 0x24e
eeprom (0x0e) ;OPNING: 0x24f
eeprom (0x52) ;OPNING: 0x250
eeprom (0x25) ;OPNING: 0x251
```

```
eeprom (0x0d) ;OPNING: 0x252
eeprom (0x45) ;OPNING: 0x253
eeprom (0x35) ;OPNING: 0x254
eeprom (0x04) ;OPNING: 0x255
eeprom (0x55) ;OPNING: 0x256
eeprom (0x22) ;OPNING: 0x257
eeprom (0x06) ;OPNING: 0x258
eeprom (0x43) ;OPNING: 0x259
eeprom (0x33) ;OPNING: 0x25a
eeprom (0x0f) ;OPNING: 0x25b
eeprom (0xcc) ;OPNING: 0x25c
```

Appendix D – C Source Code for PICAXE 20X2

```
//*****************************************************************
//
//  Kim-1 MicroChess (c) 1976-2005 Peter Jennings, www.benlo.com
//  6502 emulation   (c) 2005 Bill Forster
//  20X2 emulation   (c) 2015 Ian Mitchell
//
//  Runs an emulation of the Kim-1 Microchess on the PICAXE 20X2
//  microcontroller. Based on an idea from Bill Forster to emulate
//  6502 microprocessor instructions in C. This program generates
//  the file 20X2Microchess.bas which can be uploaded to a 20X2.
//
//*****************************************************************

// All rights reserved.

// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
// 1. Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
// 2. Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
// 3. The name of the author may not be used to endorse or promote products
//    derived from this software without specific prior written permission.

// THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS OR
// IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
// OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
// IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
// NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES// LOSS OF USE,
// DATA, OR PROFITS// OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
// THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
// THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

typedef unsigned char byte;
static FILE *f = NULL;
static int subnum = -1;

// 6502 emulation macros - register moves
#define T(src,dst) r();fprintf(f,"%s = %s : reg_f = %s\r\n",dst,src,src)
#define A "reg_a"
#define S "reg_s"
#define X "reg_x"
#define Y "reg_y"
#define TYA T(Y,A)
#define TAX T(A,X)
#define TAY T(A,Y)
#define TXA T(X,A)
#define TSX r();fprintf(f,"reg_x = ptr\r\n")
#define TXS r();fprintf(f,"ptr = reg_x\r\n")
```

```
// 6502 emulation macros - branches
#define BEQ(label) r();fprintf(f,"if reg_f=0 then goto %s\r\n",label)
#define BNE(label) r();fprintf(f,"if reg_f!=0 then goto %s\r\n",label)
#define BPL(label) r();fprintf(f,"if reg_f<0x80 then goto %s\r\n",label)
#define BMI(label) r();fprintf(f,"if reg_f>=0x80 then goto %s\r\n",label)
#define BCC(label) r();fprintf(f,"if reg_cy=0 then goto %s\r\n",label)
#define BCS(label) r();fprintf(f,"if reg_cy!=0 then goto %s\r\n",label)
#define BVC(label) r();fprintf(f,"if reg_v=0 then goto %s\r\n",label)
#define BVS(label) r();fprintf(f,"if reg_v!=0 then goto %s\r\n",label)
#define BRA(label) r();fprintf(f,"goto %s\r\n",label)
#define JEQ(label) r();fprintf(f,"if reg_f=0 then goto %s\r\n",label)
#define JMP(label) r();fprintf(f,"goto %s\r\n",label)
#define JSR(func)  r();fprintf(f,"@ptrdec = %d : goto %s\r\n",++subnum,func)
#define JSR_(func) r();fprintf(f,"gosub %s\r\n",func)
#define RTS        r();fprintf(f,"goto __return__\r\n")
#define RTS_       r();fprintf(f,"return\r\n")

// 6502 emulation macros - load registers
//  Addressing conventions;
//   default addressing mode is zero page, else indicate with suffix;
//   i = immediate
//   x = indexed, zero page
//   f = indexed, not zero page (f for "far")
#define LDAi(dat8)       r();fprintf(f,"reg_a = 0x%02x : reg_f = reg_a\r\n",dat8)
#define LDAi_(dat8)      r();fprintf(f,"reg_a = 0x%02x\r\n",dat8)
#define LDAx(addr8,idx)  r();if(addr8==0)fprintf(f,"get %s,reg_a : reg_f =
reg_a\r\n",idx);else \
   fprintf(f,"reg_f = %s+0x%02x : get reg_f,reg_a : reg_f = reg_a\r\n",idx,addr8)
#define LDAf(addr16,idx) r();fprintf(f,"reg_f = %s+0x%02x : peek reg_f,reg_a :
reg_f = reg_a\r\n",idx,addr16)
#define LDA(addr8)       r();fprintf(f,"get 0x%02x,reg_a : reg_f =
reg_a\r\n",addr8)
#define LDA_(addr8)      r();fprintf(f,"get 0x%02x,reg_a\r\n",addr8)
#define LDXi(dat8)       r();fprintf(f,"reg_x = 0x%02x : reg_f = reg_x\r\n",dat8)
#define LDXi_(dat8)      r();fprintf(f,"reg_x = 0x%02x\r\n",dat8)
#define LDX(addr8)       r();fprintf(f,"get 0x%02x,reg_x : reg_f =
reg_x\r\n",addr8)
#define LDYi(dat8)       r();fprintf(f,"reg_y = 0x%02x : reg_f = reg_y\r\n",dat8)
#define LDY(addr8)       r();fprintf(f,"get 0x%02x,reg_y : reg_f =
reg_y\r\n",addr8)
#define LDYx(addr8,idx)  r();if(addr8==0)fprintf(f,"get %s,reg_y : reg_f =
reg_y\r\n",idx);else \
   fprintf(f,"reg_f = %s+0x%02x : get reg_f,reg_y : reg_f = reg_y\r\n",idx,addr8)
#define LDYx_(addr8,idx) r();fprintf(f,"reg_f = %s+0x%02x : get
reg_f,reg_y\r\n",idx,addr8)

// 6502 emulation macros - store registers
#define STA(addr8)       r();fprintf(f,"put 0x%02x,reg_a\r\n",addr8)
#define STAx(addr8,idx)  r();if(addr8==0)fprintf(f,"put %s,reg_a\r\n",idx);else \
   fprintf(f,"temp = %s+0x%02x : put temp,reg_a\r\n",idx,addr8)
#define STX(addr8)       r();fprintf(f,"put 0x%02x,reg_x\r\n",addr8)
#define STY(addr8)       r();fprintf(f,"put 0x%02x,reg_y\r\n",addr8)
#define STYx(addr8,idx)  r();if(addr8==0)fprintf(f,"put %s,reg_y\r\n",idx);else \
   fprintf(f,"temp = %s+0x%02x : put temp,reg_y\r\n",idx,addr8)

// 6502 emulation macros - set/clear flags
#define CLD              // luckily CPU's BCD flag is cleared then never set
#define CLC              r();fprintf(f,"reg_cy = 0\r\n");
#define SEC              r();fprintf(f,"reg_cy = 1\r\n");
```

```
#define CLV             r();fprintf(f,"reg_v = 0\r\n");
#define SEV /*extra*/   r();fprintf(f,"reg_v = 1\r\n"); /*avoid problematic V
emulation*/

// 6502 emulation macros - accumulator logical operations
#define ANDi(dat8)      r();fprintf(f,"reg_a = reg_a&0x%02x : reg_f =
reg_a\r\n",dat8)
#define ANDi_(dat8)     r();fprintf(f,"reg_a = reg_a&0x%02x\r\n",dat8)
#define ORA(addr8)      r();fprintf(f,"get 0x%02x,reg_f : reg_a = reg_a|reg_f :
reg_f = reg_a\r\n",addr8)
#define ORA_(addr8)     r();fprintf(f,"get 0x%02x,reg_f : reg_a =
reg_a|reg_f\r\n",addr8)

// 6502 emulation macros - shifts and rotates
#define ASL(addr8)      r();fprintf(f,"get 0x%02x,reg_f : reg_cy = reg_f>>7 :
reg_f = reg_f<<1 : put 0x%02x,reg_f\r\n",addr8,addr8)
#define ROL(addr8)      r();fprintf(f,"get 0x%02x,reg_f : temp = reg_f>>7 : reg_f
= reg_f<<1|reg_cy : put 0x%02x,reg_f : reg_cy = temp\r\n",addr8,addr8)
#define LSR             r();fprintf(f,"reg_cy = reg_a&0x01 : reg_a = reg_a>>1 :
reg_f = reg_a\r\n")

// 6502 emulation macros - push and pull
#define PHA             r();fprintf(f,"@ptrdec = reg_a\r\n")
#define PLA             r();fprintf(f,"inc ptr : reg_a = @ptr\r\n")
#define PHY             r();fprintf(f,"@ptrdec = reg_y\r\n")
#define PLY             r();fprintf(f,"inc ptr : reg_y = @ptr\r\n")
#define PHP             r();fprintf(f,"temp = reg_v<<1|reg_cy : temp =
reg_f>>7<<3|temp : temp = reg_f max 1<<2|temp : @ptrdec = temp\r\n")
#define PLP             r();fprintf(f,"inc ptr : temp = @ptr : reg_f =
temp<<4&0xc0: reg_cy = temp&0x01 : reg_v = temp>>1&0x01\r\n")

// 6502 emulation macros - compare
// use ones complement plus one to get the correct difference and carry out
#define CMPi(dat8)      r();fprintf(f,"reg_fc = reg_a+0x%02x ;
CMPi(0x%02x)\r\n",(~dat8+1)&0xff,dat8)
#define CMP(addr8)      r();fprintf(f,"get 0x%02x,reg_f : reg_f = not reg_f :
reg_fc = reg_a+reg_f+1\r\n",addr8)
#define CMPx(addr8,idx)  r();if(addr8==0)fprintf(f,"get %s,reg_f : reg_f = not
reg_f : reg_fc = reg_a+reg_f+1\r\n",idx);else \
   fprintf(f,"reg_f = %s+0x%02x : get reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1\r\n",idx,addr8)
#define CMPf(addr16,idx) r();fprintf(f,"reg_f = %s+0x%02x : peek reg_f,reg_f :
reg_f = not reg_f : reg_fc = reg_a+reg_f+1\r\n",idx,addr16)
#define CPXi(dat8)      r();fprintf(f,"reg_fc = reg_x+0x%02x ;
CPXi(0x%02x)\r\n",(~dat8+1)&0xff,dat8)
#define CPXf(addr16,idx) r();fprintf(f,"reg_f = %s+0x%02x : peek reg_f,reg_f :
reg_f = not reg_f : reg_fc = reg_x+reg_f+1\r\n",idx,addr16)
#define CPYi(dat8)      r();fprintf(f,"reg_fc = reg_y+0x%02x ;
CPYi(0x%02x)\r\n",(~dat8+1)&0xff,dat8)

// 6502 emulation macros - increment,decrement
#define DEX             r();fprintf(f,"dec reg_x : reg_f = reg_x\r\n")
#define DEY             r();fprintf(f,"dec reg_y : reg_f = reg_y\r\n")
#define DEC(addr8)      r();fprintf(f,"get 0x%02x,reg_f : dec reg_f : put
0x%02x,reg_f\r\n",addr8,addr8)
#define INX             r();fprintf(f,"inc reg_x : reg_f = reg_x\r\n")
#define INY             r();fprintf(f,"inc reg_y : reg_f = reg_y\r\n")
#define INC(addr8)      r();fprintf(f,"get 0x%02x,reg_f : inc reg_f : put
0x%02x,reg_f\r\n",addr8,addr8)
```

```
#define INCx(addr8,idx)  r();fprintf(f,"temp = %s+0x%02x : get temp,reg_f : inc
reg_f : put temp,reg_f\r\n",idx,addr8)

// 6502 emulation macros - add
#define ADCi(dat8)       r();fprintf(f,"reg_fc = reg_a+0x%02x+reg_cy : reg_a =
reg_f\r\n",dat8)
#define ADC(addr8)       r();fprintf(f,"get 0x%02x,reg_f : reg_fc =
reg_a+reg_f+reg_cy : reg_a = reg_f\r\n",addr8)
#define ADCx(addr8,idx)  r();fprintf(f,"reg_f = %s+0x%02x : get reg_f,reg_f :
reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f\r\n",idx,addr8)
#define ADCf(addr16,idx) r();fprintf(f,"reg_f = %s+0x%02x : peek reg_f,reg_f :
reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f\r\n",idx,addr16)

// 6502 emulation macros - subtraction
//   (note that using ones complement both as an input and an output
//   the carry flag has opposite sense to that used for adc)
#define SBC(addr8)       r();fprintf(f,"get 0x%02x,reg_f : reg_f = not reg_f :
reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f\r\n",addr8)
#define SBCx(addr8,idx) r();if(addr8==0)fprintf(f,"get %s,reg_f : reg_f = not
reg_f: reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f\r\n",idx);else \
   fprintf(f,"reg_f = %s+0x%02x : get reg_f,reg_f : reg_f = not reg_f: reg_fc =
reg_a+reg_f+reg_cy : reg_a = reg_f\r\n",idx,addr8)

// page zero variables
static const byte BOARD  = 0x00;
static const byte BK     = 0x10;
static const byte LEVEL1 = 0x20; // allow change level from menu
static const byte LEVEL2 = 0x21; // allow change level from menu
static const byte PIECE  = 0x22;
static const byte SQUARE = 0x23;
static const byte SP2    = 0x24;
static const byte SP1    = 0x25;
static const byte INCHEK = 0x26;
static const byte STATE  = 0x27;
static const byte MOVEN  = 0x28;
static const byte REV    = 0x29;
static const byte OMOVE  = 0x2A;
static const byte WCAP0  = 0x2B;
static const byte COUNT  = 0x2C;
static const byte BCAP2  = 0x2C;
static const byte WCAP2  = 0x2D;
static const byte BCAP1  = 0x2E;
static const byte WCAP1  = 0x2F;
static const byte BCAP0  = 0x30;
static const byte MOB    = 0x31;
static const byte MAXC   = 0x32;
static const byte CC     = 0x33;
static const byte PCAP   = 0x34;
static const byte BMOB   = 0x31;
static const byte BMAXC  = 0x32;
static const byte BMCC   = 0x33; // was BCC, make sure not confused with
instruction definition
static const byte BMAXP  = 0x34;
static const byte XMAXC  = 0x36;
static const byte WMOB   = 0x39;
static const byte WMAXC  = 0x3A;
static const byte WCC    = 0x3B;
static const byte WMAXP  = 0x3C;
static const byte PMOB   = 0x3D;
```

```
static const byte PMAXC  = 0x3E;
static const byte PCC    = 0x3F;
static const byte PCP    = 0x40;
static const byte BESTM  = 0x41;
static const byte BESTV  = 0x42;
static const byte BESTP  = 0x43;
static const byte DIS3   = 0x41;
static const byte DIS2   = 0x42;
static const byte DIS1   = 0x43;
static const byte temp   = 0x44; // force MOVE to here instead of 0xFF

static const byte SETW_data[] =
{
   0x03, 0x04, 0x00, 0x07, 0x02, 0x05, 0x01, 0x06,
   0x10, 0x17, 0x11, 0x16, 0x12, 0x15, 0x14, 0x13,
   0x73, 0x74, 0x70, 0x77, 0x72, 0x75, 0x71, 0x76,
   0x60, 0x67, 0x61, 0x66, 0x62, 0x65, 0x64, 0x63
};
static const byte MOVEX_data[]  =
{
   0x00, 0xF0, 0xFF, 0x01, 0x10, 0x11, 0x0F, 0xEF, 0xF1,
   0xDF, 0xE1, 0xEE, 0xF2, 0x12, 0x0E, 0x1F, 0x21
};
static const byte POINTS_data[] =
{
   0x0B, 0x0A, 0x06, 0x06, 0x04, 0x04, 0x04, 0x04,
   0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02
};
static const byte OPNING_data[] =
{
   0x99, 0x25, 0x0B, 0x25, 0x01, 0x00, 0x33, 0x25,
   0x07, 0x36, 0x34, 0x0D, 0x34, 0x34, 0x0E, 0x52,
   0x25, 0x0D, 0x45, 0x35, 0x04, 0x55, 0x22, 0x06,
   0x43, 0x33, 0x0F, 0xCC
};
static const unsigned int SETW = 0x10;
static const unsigned int MOVEX = sizeof(SETW_data)+SETW;
static const unsigned int POINTS = sizeof(MOVEX_data)+MOVEX;
static const unsigned int OPNING = sizeof(POINTS_data)+POINTS;

// label destinations
#define RESTART_CHESS_ fprintf(f,"_RESTART_CHESS_:\r\n");
#define CHESS_BEGIN_   fprintf(f,"_CHESS_BEGIN_:\r\n");
#define INITCLEAR_     fprintf(f,"_INITCLEAR_:\r\n");
#define WHSET_         fprintf(f,"_WHSET_:\r\n");
#define NOSET_         fprintf(f,"_NOSET_:\r\n");
#define NOREV_         fprintf(f,"_NOREV_:\r\n");
#define CLDSP_         fprintf(f,"_CLDSP_:\r\n");
#define CLDSP2_        fprintf(f,"_CLDSP2_:\r\n");
#define NOGO_          fprintf(f,"_NOGO_:\r\n");
#define NOMV_          fprintf(f,"_NOMV_:\r\n");
#define DONE_          fprintf(f,"_DONE_:\r\n");
#define JANUS_         fprintf(f,"_JANUS_:\r\n");
#define OVER_          fprintf(f,"_OVER_:\r\n");
#define NOQ_           fprintf(f,"_NOQ_:\r\n");
#define ELOOP_         fprintf(f,"_ELOOP_:\r\n");
#define FOUN_          fprintf(f,"_FOUN_:\r\n");
#define LESS_          fprintf(f,"_LESS_:\r\n");
#define NOCAP_         fprintf(f,"_NOCAP_:\r\n");
```

```
#define XRT_          fprintf(f,"_XRT_:\r\n");
#define ON4_          fprintf(f,"_ON4_:\r\n");
#define NOCOUNT_      fprintf(f,"_NOCOUNT_:\r\n");
#define RETJ_         fprintf(f,"_RETJ_:\r\n");
#define TREE_         fprintf(f,"_TREE_:\r\n");
#define LOOPX_        fprintf(f,"_LOOPX_:\r\n");
#define FOUNX_        fprintf(f,"_FOUNX_:\r\n");
#define NOMAX_        fprintf(f,"_NOMAX_:\r\n");
#define UPTREE_       fprintf(f,"_UPTREE_:\r\n");
#define INPUT_        fprintf(f,"_INPUT_:\r\n");
#define ERROR_        fprintf(f,"_ERROR_:\r\n");
#define DISP_         fprintf(f,"_DISP_:\r\n");
#define SEARCH_       fprintf(f,"_SEARCH_:\r\n");
#define HERE_         fprintf(f,"_HERE_:\r\n");
#define GNMZ_         fprintf(f,"_GNMZ_:\r\n");
#define GNMX_         fprintf(f,"_GNMX_:\r\n");
#define CLEAR_        fprintf(f,"_CLEAR_:\r\n");
#define GNM_          fprintf(f,"_GNM_:\r\n");
#define NEWP_         fprintf(f,"_NEWP_:\r\n");
#define NEX_          fprintf(f,"_NEX_:\r\n");
#define KING_         fprintf(f,"_KING_:\r\n");
#define QUEEN_        fprintf(f,"_QUEEN_:\r\n");
#define ROOK_         fprintf(f,"_ROOK_:\r\n");
#define AGNR_         fprintf(f,"_AGNR_:\r\n");
#define BISHOP_       fprintf(f,"_BISHOP_:\r\n");
#define KNIGHT_       fprintf(f,"_KNIGHT_:\r\n");
#define AGNN_         fprintf(f,"_AGNN_:\r\n");
#define PAWN_         fprintf(f,"_PAWN_:\r\n");
#define P1_           fprintf(f,"_P1_:\r\n");
#define P2_           fprintf(f,"_P2_:\r\n");
#define P3_           fprintf(f,"_P3_:\r\n");
#define SNGMV_        fprintf(f,"_SNGMV_:\r\n");
#define ILL1_         fprintf(f,"_ILL1_:\r\n");
#define LINE_         fprintf(f,"_LINE_:\r\n");
#define OVL_          fprintf(f,"_OVL_:\r\n");
#define ILL_          fprintf(f,"_ILL_:\r\n");
#define REVERSE_      fprintf(f,"_REVERSE_:\r\n");
#define ETC_          fprintf(f,"_ETC_:\r\n");
#define CMOVE_        fprintf(f,"_CMOVE_:\r\n");
#define LOOP_         fprintf(f,"_LOOP_:\r\n");
#define NO_           fprintf(f,"_NO_:\r\n");
#define SPX_          fprintf(f,"_SPX_:\r\n");
#define RETL_         fprintf(f,"_RETL_:\r\n");
#define ILLEGAL_      fprintf(f,"_ILLEGAL_:\r\n");
#define RESET_        fprintf(f,"_RESET_:\r\n");
#define GENRM_        fprintf(f,"_GENRM_:\r\n");
#define RUM_          fprintf(f,"_RUM_:\r\n");
#define UMOVE_        fprintf(f,"_UMOVE_:\r\n");
#define MOVE_         fprintf(f,"_MOVE_:\r\n");
#define CHECK_        fprintf(f,"_CHECK_:\r\n");
#define TAKE_         fprintf(f,"_TAKE_:\r\n");
#define STRV_         fprintf(f,"_STRV_:\r\n");
#define CKMATE_       fprintf(f,"_CKMATE_:\r\n");
#define NOCHEK_       fprintf(f,"_NOCHEK_:\r\n");
#define RETV_         fprintf(f,"_RETV_:\r\n");
#define RETP_         fprintf(f,"_RETP_:\r\n");
#define GO_           fprintf(f,"_GO_:\r\n");
#define END_          fprintf(f,"_END_:\r\n");
#define NOOPEN_       fprintf(f,"_NOOPEN_:\r\n");
```

```
#define MV2_           fprintf(f,"_MV2_:\r\n");
#define MATE_          fprintf(f,"_MATE_:\r\n");
#define DISMV_         fprintf(f,"_DISMV_:\r\n");
#define DROL_          fprintf(f,"_DROL_:\r\n");
#define STRATGY_       fprintf(f,"_STRATGY_:\r\n");
#define POS_           fprintf(f,"_POS_:\r\n");
#define POSN_          fprintf(f,"_POSN_:\r\n");
#define NOPOSN_        fprintf(f,"_NOPOSN_:\r\n");
#define POUT_          fprintf(f,"_POUT_:\r\n");
#define KIN_           fprintf(f,"_KIN_:\r\n");
#define SETUP_         fprintf(f,"_SETUP_:\r\n");
#define TESTLEVEL1_    fprintf(f,"_TESTLEVEL1_:\r\n");
#define TESTLEVEL2_    fprintf(f,"_TESTLEVEL2_:\r\n");
#define TESTLEVEL3_    fprintf(f,"_TESTLEVEL3_:\r\n");
#define TESTSAVE_      fprintf(f,"_TESTSAVE_:\r\n");
#define TESTLOAD_      fprintf(f,"_TESTLOAD_:\r\n");
#define TESTRESTORE_   fprintf(f,"_TESTRESTORE_:\r\n");
#define TESTUNDO_      fprintf(f,"_TESTUNDO_:\r\n");

// labels
#define RESTART_CHESS  "_RESTART_CHESS_"
#define CHESS_BEGIN    "_CHESS_BEGIN_"
#define INITCLEAR      "_INITCLEAR_"
#define WHSET          "_WHSET_"
#define NOSET          "_NOSET_"
#define NOREV          "_NOREV_"
#define CLDSP          "_CLDSP_"
#define CLDSP2         "_CLDSP2_"
#define NOGO           "_NOGO_"
#define NOMV           "_NOMV_"
#define DONE           "_DONE_"
#define JANUS          "_JANUS_"
#define OVER           "_OVER_"
#define NOQ            "_NOQ_"
#define ELOOP          "_ELOOP_"
#define FOUN           "_FOUN_"
#define LESS           "_LESS_"
#define NOCAP          "_NOCAP_"
#define XRT            "_XRT_"
#define ON4            "_ON4_"
#define NOCOUNT        "_NOCOUNT_"
#define RETJ           "_RETJ_"
#define TREE           "_TREE_"
#define LOOPX          "_LOOPX_"
#define FOUNX          "_FOUNX_"
#define NOMAX          "_NOMAX_"
#define UPTREE         "_UPTREE_"
#define INPUT          "_INPUT_"
#define ERROR          "_ERROR_"
#define DISP           "_DISP_"
#define SEARCH         "_SEARCH_"
#define HERE           "_HERE_"
#define GNMZ           "_GNMZ_"
#define GNMX           "_GNMX_"
#define CLEAR          "_CLEAR_"
#define GNM            "_GNM_"
#define NEWP           "_NEWP_"
#define NEX            "_NEX_"
#define KING           "_KING_"
```

```
#define QUEEN          "_QUEEN_"
#define ROOK           "_ROOK_"
#define AGNR           "_AGNR_"
#define BISHOP         "_BISHOP_"
#define KNIGHT         "_KNIGHT_"
#define AGNN           "_AGNN_"
#define PAWN           "_PAWN_"
#define P1             "_P1_"
#define P2             "_P2_"
#define P3             "_P3_"
#define SNGMV          "_SNGMV_"
#define ILL1           "_ILL1_"
#define LINE           "_LINE_"
#define OVL            "_OVL_"
#define ILL            "_ILL_"
#define REVERSE        "_REVERSE_"
#define ETC            "_ETC_"
#define CMOVE          "_CMOVE_"
#define LOOP           "_LOOP_"
#define NO             "_NO_"
#define SPX            "_SPX_"
#define RETL           "_RETL_"
#define ILLEGAL        "_ILLEGAL_"
#define RESET          "_RESET_"
#define GENRM          "_GENRM_"
#define RUM            "_RUM_"
#define UMOVE          "_UMOVE_"
#define MOVE           "_MOVE_"
#define CHECK          "_CHECK_"
#define TAKE           "_TAKE_"
#define STRV           "_STRV_"
#define CKMATE         "_CKMATE_"
#define NOCHEK         "_NOCHEK_"
#define RETV           "_RETV_"
#define RETP           "_RETP_"
#define GO             "_GO_"
#define END            "_END_"
#define NOOPEN         "_NOOPEN_"
#define MV2            "_MV2_"
#define MATE           "_MATE_"
#define DISMV          "_DISMV_"
#define DROL           "_DROL_"
#define STRATGY        "_STRATGY_"
#define POS            "_POS_"
#define POSN           "_POSN_"
#define NOPOSN         "_NOPOSN_"
#define POUT           "_POUT_"
#define KIN            "_KIN_"
#define SETUP          "_SETUP_"
#define TESTLEVEL1     "_TESTLEVEL1_"
#define TESTLEVEL2     "_TESTLEVEL2_"
#define TESTLEVEL3     "_TESTLEVEL3_"
#define TESTSAVE       "_TESTSAVE_"
#define TESTLOAD       "_TESTLOAD_"
#define TESTRESTORE    "_TESTRESTORE_"
#define TESTUNDO       "_TESTUNDO_"

static void r(void)
{
```

```
   // this function generates the labels for the __return__ subroutine
   // if subnum has been incremented then this is remembered and a
   // label is generated, otherwise just padding
   // subnum is incremented from JSR
   static int testsubnum = -1;
   if (subnum==testsubnum)
   {
      fprintf(f,"      ");
      return;
   }
   testsubnum = subnum;
   fprintf(f,"_%02d:  ",subnum);
}

static void copyright(void)
{
   fprintf(f,"#rem\r\n");

fprintf(f,"*****************************************************************
\r\n");

fprintf(f,"*****************************************************************
\r\n");

fprintf(f,"*****************************************************************
\r\n");

fprintf(f,"*****************************************************************
\r\n");

fprintf(f,"*****************************************************************
\r\n");
   fprintf(f,"\r\n");
   fprintf(f,"  Kim-1 MicroChess (c) 1976-2005 Peter Jennings,
www.benlo.com\r\n");
   fprintf(f,"  6502 emulation   (c) 2005 Bill Forster\r\n");
   fprintf(f,"  20X2 emulation   (c) 2015 Ian Mitchell\r\n");
   fprintf(f,"\r\n");
   fprintf(f,"  Runs an emulation of the Kim-1 Microchess on the PICAXE
20X2\r\n");
   fprintf(f,"  microcontroller. Based on an idea from Bill Forster to
emulate\r\n");
   fprintf(f,"  6502 microprocessor instructions in C. The program is
created\r\n");
   fprintf(f,"  by running 20X2Microchess.exe. This file
(20X2Microchess.bas)\r\n");
   fprintf(f,"  is generated and can be uploaded to a 20X2.\r\n");
   fprintf(f,"\r\n");

fprintf(f,"*****************************************************************
\r\n");

fprintf(f,"*****************************************************************
\r\n");

fprintf(f,"*****************************************************************
\r\n");
```

```
fprintf(f,"*************************************************************************
\r\n");

fprintf(f,"*************************************************************************
\r\n");
   fprintf(f,"\r\n");
   fprintf(f," All rights reserved.\r\n");
   fprintf(f,"\r\n");
   fprintf(f," Redistribution and use in source and binary forms, with or
without\r\n");
   fprintf(f," modification, are permitted provided that the following
conditions\r\n");
   fprintf(f," are met:\r\n");
   fprintf(f," 1. Redistributions of source code must retain the above
copyright\r\n");
   fprintf(f,"       notice, this list of conditions and the following
disclaimer.\r\n");
   fprintf(f," 2. Redistributions in binary form must reproduce the above
copyright\r\n");
   fprintf(f,"       notice, this list of conditions and the following disclaimer
in the\r\n");
   fprintf(f,"       documentation and/or other materials provided with the
distribution.\r\n");
   fprintf(f," 3. The name of the author may not be used to endorse or promote
products\r\n");
   fprintf(f,"       derived from this software without specific prior written
permission.\r\n");
   fprintf(f,"\r\n");
   fprintf(f," THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS
OR\r\n");
   fprintf(f," IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES\r\n");
   fprintf(f," OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED.\r\n");
   fprintf(f," IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT,
INDIRECT,\r\n");
   fprintf(f," INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT\r\n");
   fprintf(f," NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES LOSS OF
USE,\r\n");
   fprintf(f," DATA, OR PROFITS OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY\r\n");
   fprintf(f," THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT\r\n");
   fprintf(f," (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF\r\n");
   fprintf(f," THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.\r\n");
   fprintf(f,"#endrem\r\n");
   fprintf(f,"\r\n");
}

static void init(void)
{
//   errno_t e = fopen_s(&f,"20X2Microchess.bas","wb");
//   if (e!=0)
   f = fopen("20X2Microchess.bas","wb");
   if (f==NULL)
```

```
   {
      printf("could not open file\r\n");
      exit(1);
   }
   copyright();
   fprintf(f,"#picaxe 20x2\r\n");
   fprintf(f,"setfreq m64\r\n");
   fprintf(f,"symbol reg_a = b0\r\n");
   fprintf(f,"symbol reg_x = b1\r\n");
   fprintf(f,"symbol reg_y = b2\r\n");
   fprintf(f,"symbol reg_v = b3\r\n");
   fprintf(f,"symbol reg_f = b4\r\n");
   fprintf(f,"symbol reg_cy = b5\r\n");
   fprintf(f,"symbol reg_fc = w2\r\n");
   fprintf(f,"symbol temp = b6\r\n");
   fprintf(f,"\r\n");
   fprintf(f,"symbol _row = b7\r\n");
   fprintf(f,"symbol _col = b8\r\n");
   fprintf(f,"symbol _loc = b9\r\n");
   fprintf(f,"symbol _pindex = b10\r\n");
   fprintf(f,"symbol _reverse = b11\r\n");
   fprintf(f,"symbol _p = b12\r\n");
   fprintf(f,"\r\n");
   fprintf(f,"      gosub __read_static_data__\r\n");
}

static void done(void)
{
   fprintf(f,"__hexbyte__:\r\n");
   fprintf(f,"      temp = reg_a>>4+\"0\"\r\n");
   fprintf(f,"      gosub __nybble__\r\n");
   fprintf(f,"      temp = reg_a&0x0f+\"0\"\r\n");
   fprintf(f,"__nybble__:\r\n");
   fprintf(f,"      if temp>\"9\" then : temp = temp+7 : endif\r\n");
   fprintf(f,"      sertxd (temp)\r\n");
   fprintf(f,"      return\r\n");
   fprintf(f,"__showboard__:\r\n");
   fprintf(f,"      gosub __backupposition__\r\n");
   fprintf(f,"      sertxd (cr,lf)\r\n");
   fprintf(f,"      gosub __rownum__\r\n");
   fprintf(f,"      gosub __line__\r\n");
   fprintf(f,"      get 0x%02x,_reverse\r\n",REV);
   fprintf(f,"      for _row=0 to 7\r\n");
   fprintf(f,"         sertxd (#_row,\"0|\")\r\n");
   fprintf(f,"         for _col = 0 to 7\r\n");
   fprintf(f,"            _loc = _row<<4+_col\r\n");
   fprintf(f,"            for _pindex = 0 to 0x1f\r\n");
   fprintf(f,"               get _pindex,_p\r\n");
   fprintf(f,"               if _p=_loc then\r\n");
   fprintf(f,"                  _p = _pindex>>4^_reverse\r\n");
   fprintf(f,"                  if _p=0 then sertxd (\"W\") : else : sertxd
(\"B\") : endif\r\n");
   fprintf(f,"                  _p = _pindex&0x0f\r\n");
   fprintf(f,"                  lookup _p,(\"KQRRBBNNPPPPPPPP\"),_p\r\n");
   fprintf(f,"                  sertxd (_p)\r\n");
   fprintf(f,"                  goto __next_location__\r\n");
   fprintf(f,"               endif\r\n");
   fprintf(f,"            next\r\n");
   fprintf(f,"            ; not found\r\n");
```

```
   fprintf(f,"            _p = _row^_col&1\r\n");
   fprintf(f,"            if _p=1 then : sertxd (\"**\") : else : sertxd (\"  \")
: endif\r\n");
   fprintf(f,"            __next_location__:\r\n");
   fprintf(f,"            sertxd (\"|\")\r\n");
   fprintf(f,"         next\r\n");
   fprintf(f,"         sertxd (#_row,\"0\",cr,lf)\r\n");
   fprintf(f,"         gosub __line__\r\n");
   fprintf(f,"      next\r\n");
   fprintf(f,"      gosub __rownum__\r\n");
   fprintf(f,"      return\r\n");
   fprintf(f,"__line__:\r\n");
   fprintf(f,"      sertxd (\"  \") : for _p=1 to 25 : sertxd (\"-\") : next :
sertxd (cr,lf)\r\n");
   fprintf(f,"      return\r\n");
   fprintf(f,"__rownum__:\r\n");
   fprintf(f,"      sertxd (\"  \") : for _p=0 to 7 : sertxd (\" 0\",#_p) : next :
sertxd (cr,lf)\r\n");
   fprintf(f,"      return\r\n");

   // emulate RTS
   fprintf(f,"__return__:\r\n");
   fprintf(f,"      inc ptr\r\n");
   fprintf(f,"      branch @ptr,(");
   for (int i=0;i<=subnum;i++)
   {
      if (i>0)
      {
         fprintf(f,",");
      }
      fprintf(f,"_%02d",i);
   }
   fprintf(f,")\r\n");


   // save board
   fprintf(f,"__saveposition__:\r\n");
   fprintf(f,"      write 0xff,0xff\r\n");
   fprintf(f,"      _loc = 0x64\r\n");

   // save position
   fprintf(f,"__saveposition0__:\r\n");
   fprintf(f,"      ;_loc has eeprom address\r\n");
   fprintf(f,"      for temp = 0 to 0x1f\r\n");
   fprintf(f,"         get temp,_p\r\n");
   fprintf(f,"         _pindex =_loc+temp\r\n");
   fprintf(f,"         write _pindex,_p\r\n");
   fprintf(f,"      next\r\n");
   fprintf(f,"      get 0x%02x,_p\r\n",REV);
   fprintf(f,"      inc _pindex\r\n");
   fprintf(f,"      write _pindex,_p\r\n");
   fprintf(f,"      get 0x%02x,_p\r\n",LEVEL1);
   fprintf(f,"      inc _pindex\r\n");
   fprintf(f,"      write _pindex,_p\r\n");
   fprintf(f,"      get 0x%02x,_p\r\n",LEVEL2);
   fprintf(f,"      inc _pindex\r\n");
   fprintf(f,"      write _pindex,_p\r\n");
   fprintf(f,"      return\r\n");
```

```
// load position
fprintf(f,"__loadposition0__:\r\n");
fprintf(f,"       ;_loc has eeprom address\r\n");
fprintf(f,"       for temp = 0 to 0x1f\r\n");
fprintf(f,"          _pindex =_loc+temp\r\n");
fprintf(f,"          read _pindex,_p\r\n");
fprintf(f,"          put temp,_p\r\n");
fprintf(f,"       next\r\n");
fprintf(f,"       inc _pindex\r\n");
fprintf(f,"       read _pindex,_p\r\n");
fprintf(f,"       put 0x%02x,_p\r\n",REV);
fprintf(f,"       inc _pindex\r\n");
fprintf(f,"       read _pindex,_p\r\n");
fprintf(f,"       put 0x%02x,_p\r\n",LEVEL1);
fprintf(f,"       inc _pindex\r\n");
fprintf(f,"       read _pindex,_p\r\n");
fprintf(f,"       put 0x%02x,_p\r\n",LEVEL2);
fprintf(f,"       return\r\n");

// load board
fprintf(f,"__loadposition__:\r\n");
fprintf(f,"       read 0xff,temp\r\n");
fprintf(f,"       if temp!=0xff then\r\n");
fprintf(f,"          sertxd (cr,lf,\"Save first.\",cr,lf)\r\n");
fprintf(f,"          return\r\n");
fprintf(f,"       endif\r\n");
fprintf(f,"       _loc = 0x64\r\n");
fprintf(f,"       gosub __loadposition0__\r\n");
fprintf(f,"       goto __showboard__\r\n");

// backup called after every move
fprintf(f,"__backupposition__:\r\n");
fprintf(f,"       write 0xfe,0xff\r\n");
fprintf(f,"       _loc = 0x8c\r\n");
fprintf(f,"       goto __saveposition0__\r\n");

// restore position
fprintf(f,"__restoreposition__:\r\n");
fprintf(f,"       read 0xfe,temp\r\n");
fprintf(f,"       if temp!=0xff then\r\n");
fprintf(f,"          sertxd (cr,lf,\"Can't restore.\",cr,lf)\r\n");
fprintf(f,"          return\r\n");
fprintf(f,"       endif\r\n");
fprintf(f,"       _loc = 0x8c\r\n");
fprintf(f,"       gosub __loadposition0__\r\n");
fprintf(f,"       goto __showboard__\r\n");

// called after every computer move
fprintf(f,"__dosaveposition__:\r\n");
fprintf(f,"       write 0xfd,0xff\r\n");
fprintf(f,"       _loc = 0xb4\r\n");
fprintf(f,"       goto __saveposition0__\r\n");

// undo position
fprintf(f,"__undoposition__:\r\n");
fprintf(f,"       read 0xfd,temp\r\n");
fprintf(f,"       if temp!=0xff then\r\n");
fprintf(f,"          sertxd (cr,lf,\"Can't undo.\",cr,lf)\r\n");
fprintf(f,"          return\r\n");
```

```
   fprintf(f,"      endif\r\n");
   fprintf(f,"      _loc = 0xb4\r\n");
   fprintf(f,"      gosub __loadposition0__\r\n");
   fprintf(f,"      goto __showboard__\r\n");

   // eeprom-read, table-readtable
   static const int data_size =
sizeof(SETW_data)+sizeof(MOVEX_data)+sizeof(POINTS_data)+sizeof(OPNING_data);
   fprintf(f,"__read_static_data__:\r\n");
   fprintf(f,"      for temp = 0 to 0x%02x\r\n",data_size-1);
   fprintf(f,"         read temp,_p\r\n");
   fprintf(f,"         _loc = temp+0x%02x\r\n",SETW);
   fprintf(f,"         poke _loc,_p\r\n");
   fprintf(f,"      next\r\n");
   fprintf(f,"      return\r\n");
   fprintf(f,"\r\n");
   for (int i=0;i<sizeof(SETW_data);i++)
   {
      fprintf(f,"      eeprom (0x%02x) ;SETW: 0x%02x\r\n",SETW_data[i],SETW+i);
   }
   for (int i=0;i<sizeof(MOVEX_data);i++)
   {
      fprintf(f,"      eeprom (0x%02x) ;MOVEX: 0x%02x\r\n",MOVEX_data[i],MOVEX+i);
   }
   for (int i=0;i<sizeof(POINTS_data);i++)
   {
      fprintf(f,"      eeprom (0x%02x) ;POINTS:
0x%02x\r\n",POINTS_data[i],POINTS+i);
   }
   for (int i=0;i<sizeof(OPNING_data);i++)
   {
      fprintf(f,"      eeprom (0x%02x) ;OPNING:
0x%02x\r\n",OPNING_data[i],OPNING+i);
   }
   fclose(f);
}

void chess( void )
{
               LDAi_   (0x00); // level 1
               STA     (LEVEL1);
               LDAi_   (0xFF); // level 1
               STA     (LEVEL2);
               LDAi_   (0x00);
               STA     (REV);
               LDXi_   (0x1F);             // clear board
               LDAi_   (0xCC);
INITCLEAR_     STAx    (BOARD,X);
               DEX;
               BPL     (INITCLEAR);

//
//
CHESS_BEGIN_   CLD;                        // INITIALIZE
               LDXi_   (0x7F);             // TWO STACKS
               TXS;
               LDXi_   (0x60);
               STX     (SP2);
//
```

```
//       ROUTINES TO LIGHT LED
//       DISPLAY AND GET KEY
//       FROM KEYBOARD
//
                JSR_    (POUT);             // DISPLAY AND
                JSR_    (KIN);              // GET INPUT   *** my routine waits
for a keypress
//              CMP     (OLDKY);            // KEY IN ACC  *** no need to debounce
//              BEQ     (OUT);              // (DEBOUNCE)
//              STA     (OLDKY);
                ANDi_   (0x5F);             // convert to upper
                CMPi    (0x58);             // [X] level 1 (super blitz)
                BNE     (TESTLEVEL2);
                LDAi_   (0x00);             // level 1
                STA     (LEVEL1);
                LDAi_   (0xFF);             // level 1
                STA     (LEVEL2);
                LDAi_   (0x11);             // indicate level 1
                JMP     (CLDSP2);
TESTLEVEL2_     CMPi    (0x59);             // [Y] level 2 (blitz)
                BNE     (TESTLEVEL3);
                LDAi_   (0x00);             // level 2
                STA     (LEVEL1);
                LDAi_   (0xFB);             // level 2
                STA     (LEVEL2);
                LDAi_   (0x22);             // indicate level 2
                JMP     (CLDSP2);
TESTLEVEL3_     CMPi    (0x5A);             // [Z] level 3 (normal)
                BNE     (TESTSAVE);
                LDAi_   (0x08);             // level 3
                STA     (LEVEL1);
                LDAi_   (0xFB);             // level 3
                STA     (LEVEL2);
                LDAi_   (0x33);             // indicate level 3
                JMP     (CLDSP2);
TESTSAVE_       CMPi    (0x53);             // [S] save position
                BNE     (TESTLOAD);
                JSR_    ("__saveposition__");  // save the board and the reverse
flag
                LDAi_   (0x55);             // indicate saved
                JMP     (CLDSP2);
TESTLOAD_       CMPi    (0x4C);             // [L] load saved position
                BNE     (TESTRESTORE);
                JSR_    ("__loadposition__");  // load the board and the reverse
flag
                LDAi_   (0x88);             // indicate loaded
                JMP     (CLDSP2);
TESTRESTORE_    CMPi    (0x52);             // [R] load saved position
                BNE     (TESTUNDO);
                JSR_    ("__restoreposition__");  // load the board and the
reverse flag
                LDAi_   (0x88);             // indicate loaded
                JMP     (CLDSP2);
TESTUNDO_       CMPi    (0x55);             // [U] undo user move
                BNE     (SETUP);
                JSR_    ("__undoposition__");  // load the board and the reverse
flag
                LDAi_   (0x88);             // indicate loaded
                JMP     (CLDSP2);
```

```
//
SETUP_          ANDi_   (0x4F);             // MASK 0-7, AND ALPHA'S (moved from
KIN)
                CMPi    (0x43);             // [C]
                BNE     (NOSET);            // SET UP
                LDXi_   (0x1F);             // BOARD
WHSET_          LDAf    (SETW,X);           // FROM
                STAx    (BOARD,X);          // SETW
                DEX;
                BPL     (WHSET);
                LDXi_   (0x1B);             // *ADDED
                STX     (OMOVE);            // INITS TO 0xFF
                LDAi_   (0x00);             // added (igm)
                STA     (REV);              // computer plays white
                JSR_    ("__dosaveposition__"); // save for undo
                LDAi_   (0xCC);             // Display CCC
                JMP     (CLDSP);            // was BNE (igm)
//
NOSET_          CMPi    (0x45);             // [E]
                BNE     (NOREV);            // REVERSE
                JSR_    (REVERSE);          // BOARD IS
                SEC;
                LDAi_   (0x01);
                SBC     (REV);
                STA     (REV);              // TOGGLE REV FLAG
                JSR_    ("__dosaveposition__"); // save for undo
                LDAi_   (0xEE);             // IS
                JMP     (CLDSP);            // was BNE (igm)
//
NOREV_          CMPi    (0x40);             // [P] (P is 0x50 but masked with
0x4f)
                BNE     (NOGO);
                JSR     (GO);               // PLAY CHESS
                JSR_    ("__dosaveposition__"); // save for undo
CLDSP_          JSR_    ("__showboard__");  // display the whole board
CLDSP2_         STA     (DIS1);             // DISPLAY
                STA     (DIS2);             // ACROSS
                STA     (DIS3);             // DISPLAY
                JMP     (CHESS_BEGIN);
//
NOGO_           CMPi    (0x0D);             // [Enter]
                BNE     (NOMV);             // MOVE MAN
                JSR     (MOVE);             // AS ENTERED
                JSR_    ("__showboard__");  // display the whole board
                JMP     (DISP);             //
NOMV_           CMPi    (0x41);             // [Q] ***Added to allow game exit***
                BEQ     (DONE);             // quit the game, exit back to system.
                JMP     (INPUT);            //
DONE_           JMP     (RESTART_CHESS);    // clean start
//
//
JANUS_          LDX     (STATE);
                BMI     (NOCOUNT);
//
//       THIS ROUTINE COUNTS OCCURRENCES
//       IT DEPENDS UPON STATE TO INDEX
//       THE CORRECT COUNTERS
//
/*COUNTS_*/     LDA     (PIECE);
```

```
                BEQ     (OVER);             // IF STATE=8
                CPXi    (0x08);             // DO NOT COUNT
                BNE     (OVER);             // BLK MAX CAP
                CMP     (BMAXP);            // MOVES FOR
                BEQ     (XRT);              // WHITE
//
OVER_           INCx    (MOB,X);            // MOBILITY
                CMPi    (0x01);             //  + QUEEN
                BNE     (NOQ);              // FOR TWO
                INCx    (MOB,X);
//
NOQ_            BVC     (NOCAP);
                LDYi    (0x0F);             // CALCULATE
                LDA     (SQUARE);           // POINTS
ELOOP_          CMPx    (BK,Y);             // CAPTURED
                BEQ     (FOUN);             // BY THIS
                DEY;                        // MOVE
                BPL     (ELOOP);
FOUN_           LDAf    (POINTS,Y);
                CMPx    (MAXC,X);
                BCC     (LESS);             // SAVE IF
                STYx    (PCAP,X);           // BEST THIS
                STAx    (MAXC,X);           // STATE
//
LESS_           CLC;
                PHP;                        // ADD TO
                ADCx    (CC,X);             // CAPTURE
                STAx    (CC,X);             // COUNTS
                PLP;
//
NOCAP_          CPXi    (0x04);
                BEQ     (ON4);
                BMI     (TREE);             //(=00 ONLY)
XRT_            RTS;
//
//      GENERATE FURTHER MOVES FOR COUNT
//      AND ANALYSIS
//
ON4_            LDA     (XMAXC);            // SAVE ACTUAL
                STA     (WCAP0);            // CAPTURE
                LDAi    (0x00);             // STATE=0
                STA     (STATE);
                JSR     (MOVE);             // GENERATE
                JSR_    (REVERSE);          // IMMEDIATE
                JSR     (GNMZ);             // REPLY MOVES
                JSR_    (REVERSE);
                LDAi    (0x08);             // STATE=8
                STA     (STATE);            // GENERATE
                JSR     (GNM);              // CONTINUATION
                JSR     (UMOVE);            // MOVES
                JMP     (STRATGY);
NOCOUNT_        CPXi    (0xF9);
                BNE     (TREE);
//
//      DETERMINE IF THE KING CAN BE
//      TAKEN, USED BY CHKCHK
//
                LDA     (BK);               // IS KING
                CMP     (SQUARE);           // IN CHECK?
```

```
                BNE     (RETJ);             // SET INCHEK=0
                LDAi    (0x00);             // IF IT IS
                STA     (INCHEK);
RETJ_           RTS;
//
//      IF A PIECE HAS BEEN CAPTURED BY
//      A TRIAL MOVE, GENERATE REPLIES &
//      EVALUATE THE EXCHANGE GAIN/LOSS
//
TREE_           BVC     (RETJ);             // NO CAP
                LDYi    (0x07);             // (PIECES)
                LDA     (SQUARE);
LOOPX_          CMPx    (BK,Y);
                BEQ     (FOUNX);
                DEY;
                BEQ     (RETJ);             // (KING)
                BPL     (LOOPX);            // SAVE
FOUNX_          LDAf    (POINTS,Y);         // BEST CAP
                CMPx    (BCAP0,X);          // AT THIS
                BCC     (NOMAX);            // LEVEL
                STAx    (BCAP0,X);
NOMAX_          DEC     (STATE);
                LDA     (LEVEL2);           // IF STATE=FB  (WRF, was LDAi
(0xFB);)
                CMP     (STATE);            // TIME TO TURN
                BEQ     (UPTREE);           // AROUND
                JSR     (GENRM);            // GENERATE FURTHER
UPTREE_         INC     (STATE);            // CAPTURES
                RTS;
//
//      THE PLAYER'S MOVE IS INPUT
//
INPUT_          CMPi    (0x08);             // NOT A LEGAL
                BCS     (ERROR);            // SQUARE #
                JSR_    (DISMV);
                JMP     (DISP);             // fall through
ERROR_          JMP     (CHESS_BEGIN);
//
// display
//
DISP_           LDXi    (0x1F);
SEARCH_         LDAx    (BOARD,X);
                CMP     (DIS2);
                BEQ     (HERE);             // DISPLAY
                DEX;                        // PIECE AT
                BPL     (SEARCH);           // FROM
HERE_           STX     (DIS1);             // SQUARE
                STX     (PIECE);
                JMP     (CHESS_BEGIN);
//
//      GENERATE ALL MOVES FOR ONE
//      SIDE, CALL JANUS AFTER EACH
//      ONE FOR NEXT STEP
//
GNMZ_           LDXi    (0x10);             // CLEAR
//
GNMX_           LDAi    (0x00);             // COUNTERS
CLEAR_          STAx    (COUNT,X);
                DEX;
```

```
                BPL     (CLEAR);
//
GNM_            LDAi    (0x10);             // SET UP
                STA     (PIECE);            // PIECE
NEWP_           DEC     (PIECE);            // NEW PIECE
                BPL     (NEX);              // ALL DONE?
                RTS;                        //    -YES
//
NEX_            JSR_    (RESET);            // READY
                LDY     (PIECE);            // GET PIECE
                LDXi    (0x08);
                STX     (MOVEN);            // COMMON START
                CPYi    (0x08);             // WHAT IS IT?
                BPL     (PAWN);             // PAWN
                CPYi    (0x06);
                BPL     (KNIGHT);           // KNIGHT
                CPYi    (0x04);
                BPL     (BISHOP);           // BISHOP
                CPYi    (0x01);
                BEQ     (QUEEN);            // QUEEN
                BPL     (ROOK);             // ROOK
//
KING_           JSR     (SNGMV);            // MUST BE KING!
                BNE     (KING);             // MOVES
                BEQ     (NEWP);             // 8 TO 1
QUEEN_          JSR     (LINE);
                BNE     (QUEEN);            // MOVES
                BEQ     (NEWP);             // 8 TO 1
//
ROOK_           LDXi    (0x04);
                STX     (MOVEN);            // MOVES
AGNR_           JSR     (LINE);             // 4 TO 1
                BNE     (AGNR);
                BEQ     (NEWP);
//
BISHOP_         JSR     (LINE);
                LDA     (MOVEN);            // MOVES
                CMPi    (0x04);             // 8 TO 5
                BNE     (BISHOP);
                BEQ     (NEWP);
//
KNIGHT_         LDXi    (0x10);
                STX     (MOVEN);            // MOVES
AGNN_           JSR     (SNGMV);            // 16 TO 9
                LDA     (MOVEN);
                CMPi    (0x08);
                BNE     (AGNN);
                BEQ     (NEWP);
//
PAWN_           LDXi    (0x06);
                STX     (MOVEN);
P1_             JSR     (CMOVE);            // RIGHT CAP?
                BVC     (P2);
                BMI     (P2);
                JSR     (JANUS);            // YES
P2_             JSR_    (RESET);
                DEC     (MOVEN);            // LEFT CAP?
                LDA     (MOVEN);
                CMPi    (0x05);
```

```
                BEQ     (P1);
P3_             JSR     (CMOVE);            // AHEAD
                BVS     (NEWP);             // ILLEGAL
                BMI     (NEWP);
                JSR     (JANUS);
                LDA     (SQUARE);           // GETS TO
                ANDi_   (0xF0);             // 3RD RANK?
                CMPi    (0x20);
                BEQ     (P3);               // DO DOUBLE
                BRA     (NEWP);             // JMP (NEWP);
//
//      CALCULATE SINGLE STEP MOVES
//      FOR K,N
//
SNGMV_          JSR     (CMOVE);            // CALC MOVE
                BMI     (ILL1);             // -IF LEGAL
                JSR     (JANUS);            // -EVALUATE
ILL1_           JSR_    (RESET);
                DEC     (MOVEN);
                RTS;
//
//     CALCULATE ALL MOVES DOWN A
//     STRAIGHT LINE FOR Q,B,R
//
LINE_           JSR     (CMOVE);            // CALC MOVE
                BCC     (OVL);              // NO CHK
                BVC     (LINE);             // NOCAP
OVL_            BMI     (ILL);              // RETURN
                PHP;
                JSR     (JANUS);            // EVALUATE POSN
                PLP;
                BVC     (LINE);             // NOT A CAP
ILL_            JSR_    (RESET);            // LINE STOPPED
                DEC     (MOVEN);            // NEXT DIR
                RTS;
//
//      EXCHANGE SIDES FOR REPLY
//      ANALYSIS
//
REVERSE_        LDXi_   (0x0F);
ETC_            SEC;
                LDYx_   (BK,X);             // SUBTRACT
                LDAi_   (0x77);             // POSITION
                SBCx    (BOARD,X);          // FROM 77
                STAx    (BK,X);
                STYx    (BOARD,X);          // AND
                SEC;
                LDAi_   (0x77);             // EXCHANGE
                SBCx    (BOARD,X);          // PIECES
                STAx    (BOARD,X);
                DEX;
                BPL     (ETC);
                RTS_;
//
//        CMOVE CALCULATES THE TO SQUARE
//        USING SQUARE AND THE MOVE
//       TABLE  FLAGS SET AS FOLLOWS_
//       N - ILLEGAL MOVE
//       V - CAPTURE (LEGAL UNLESS IN CH)
```

```
//       C - ILLEGAL BECAUSE OF CHECK
//       [MY THANKS TO JIM BUTTERFIELD
//        WHO WROTE THIS MORE EFFICIENT
//        VERSION OF CMOVE]
//
CMOVE_          LDA     (SQUARE);           // GET SQUARE
                LDX     (MOVEN);            // MOVE POINTER
                CLC;
                ADCf    (MOVEX,X);          // MOVE LIST
                STA     (SQUARE);           // NEW POS'N
                ANDi    (0x88);
                BNE     (ILLEGAL);          // OFF BOARD
                LDA     (SQUARE);
                LDXi    (0x20);
LOOP_           DEX;                        // IS TO
                BMI     (NO);               // SQUARE
                CMPx    (BOARD,X);          // OCCUPIED?
                BNE     (LOOP);
                CPXi    (0x10);             // BY SELF?
                BMI     (ILLEGAL);
//              LDAi    (0x7F);             // MUST BE CAP!
//              ADCi    (0x01);             // SET V FLAG
                SEV;    LDAi(0x80);         // Avoid problematic V emulation
                JMP     (SPX);              // (JMP, was BVS [igm])
//
NO_             CLV;                        // NO CAPTURE
//
SPX_            LDA     (STATE);            // SHOULD WE
                BMI     (RETL);             // DO THE
                CMP     (LEVEL1);           // CHECK CHECK? (WRF_ was CMPi
(0x08);)
                BPL     (RETL);
//
//       CHKCHK REVERSES SIDES
//       AND LOOKS FOR A KING
//       CAPTURE TO INDICATE
//       ILLEGAL MOVE BECAUSE OF
//       CHECK  SINCE THIS IS
//       TIME CONSUMING, IT IS NOT
//       ALWAYS DONE
//
/*CHKCHK_*/     PHA;                        // STATE
                PHP;
                LDAi    (0xF9);
                STA     (STATE);            // GENERATE
                STA     (INCHEK);           // ALL REPLY
                JSR     (MOVE);             // MOVES TO
                JSR_    (REVERSE);          // SEE IF KING
                JSR     (GNM);              // IS IN
                JSR     (RUM);              // CHECK
                PLP;
                PLA;
                STA     (STATE);
                LDA     (INCHEK);
                BMI     (RETL);             // NO - SAFE
                SEC;                        // YES - IN CHK
                LDAi    (0xFF);
                RTS;
//
```

```
RETL_           CLC;                        // LEGAL
                LDAi    (0x00);             // RETURN
                RTS;
//
ILLEGAL_        LDAi    (0xFF);
                CLC;                        // ILLEGAL
                CLV;                        // RETURN
                RTS;
//
//       REPLACE PIECE ON CORRECT SQUARE
//
RESET_          LDX     (PIECE);            // GET LOGAT
                LDAx    (BOARD,X);          // FOR PIECE
                STA     (SQUARE);           // FROM BOARD
                RTS_;
//
//
GENRM_          JSR     (MOVE);             // MAKE MOVE
                JSR_    (REVERSE);          // REVERSE BOARD
                JSR     (GNM);              // GENERATE MOVES
RUM_            JSR_    (REVERSE);          // REVERSE BACK
//
//       ROUTINE TO UNMAKE A MOVE MADE BY
//         MOVE
//
UMOVE_          TSX;                        // UNMAKE MOVE
                STX     (SP1);
                LDX     (SP2);              // EXCHANGE
                TXS;                        // STACKS
                PLA;                        // MOVEN
                STA     (MOVEN);
                PLA;                        // CAPTURED
                STA     (PIECE);            // PIECE
                TAX;
                PLA;                        // FROM SQUARE
                STAx    (BOARD,X);
                PLA;                        // PIECE
                TAX;
                PLA;                        // TO SOUARE
                STA     (SQUARE);
                STAx    (BOARD,X);
                JMP     (STRV);
//
//       THIS ROUTINE MOVES PIECE
//       TO SQUARE, PARAMETERS
//       ARE SAVED IN A STACK TO UNMAKE
//       THE MOVE LATER
//
MOVE_           TSX;
                STX     (SP1);              // SWITCH
                LDX     (SP2);              // STACKS
                TXS;
                LDA     (SQUARE);
                PHA;                        // TO SQUARE
                TAY;
                LDXi    (0x1F);
CHECK_          CMPx    (BOARD,X);          // CHECK FOR
                BEQ     (TAKE);             // CAPTURE
                DEX;
```

```
                BPL     (CHECK);
                LDXi_   (temp);             // extra instruction to avoid access
to 0xFF
                                            // (0x4F on Microchess, 0x50+0xFF),
                                            // force to temp instead (igm)
TAKE_           LDAi    (0xCC);
                STAx    (BOARD,X);
                TXA;                        // CAPTURED
                PHA;                        // PIECE
                LDX     (PIECE);
                LDAx    (BOARD,X);
                STYx    (BOARD,X);          // FROM
                PHA;                        // SQUARE
                TXA;
                PHA;                        // PIECE
                LDA     (MOVEN);
                PHA;                        // MOVEN

//
//  Fortunately when we swap stacks we jump here and swap back before
//  returning. The original code does this so we can take advantage
//  on the picaxe and implement a stack and calling/return mechanism
//  that uses sratchpad to store an ID number for each return address.
//  This allows a subroutine call depth much greater than 8 (the limit
//  of the picaxe).
//
STRV_           TSX;
                STX     (SP2);              // SWITCH
                LDX     (SP1);              // STACKS
                TXS;                        // BACK
                RTS;
//
//       CONTINUATION OF SUB STRATGY
//       -CHECKS FOR CHECK OR CHECKMATE
//       AND ASSIGNS VALUE TO MOVE
//
CKMATE_         LDX     (BMAXC);            // CAN BLK CAP
                CPXf    (POINTS,"0");       // MY KING?
                BNE     (NOCHEK);
                LDAi    (0x00);             // GULP!
                JMP     (RETV);             // DUMB MOVE! was BEQ (igm)
//
NOCHEK_         LDX     (BMOB);             // IS BLACK
                BNE     (RETV);             // UNABLE TO
                LDX     (WMAXP);            // MOVE AND
                BNE     (RETV);             // KING IN CH?
                LDAi    (0xFF);             // YES! MATE
//
RETV_           LDXi    (0x04);             // RESTORE
                STX     (STATE);            // STATE=4
//
//       THE VALUE OF THE MOVE (IN ACCU)
//       IS COMPARED TO THE BEST MOVE AND
//       REPLACES IT IF IT IS BETTER
//
/*PUSH_*/       CMP     (BESTV);            // IS THIS BEST
                BCC     (RETP);             // MOVE SO FAR?
                BEQ     (RETP);
                STA     (BESTV);            // YES!
```

```
                LDA_    (PIECE);            // SAVE IT
                STA     (BESTP);
                LDA_    (SQUARE);
                STA     (BESTM);            // FLASH DISPLAY
RETP_           LDAi_   ('.');              // print ... instead of flashing disp
                fprintf(f,"      sertxd (reg_a)\r\n");
                RTS;
//
//       MAIN PROGRAM TO PLAY CHESS
//       PLAY FROM OPENING OR THINK
//
GO_             LDX     (OMOVE);            // OPENING?
                BMI     (NOOPEN);           // -NO   *ADD CHANGE FROM BPL
                LDA     (DIS3);             // -YES WAS
                CMPf    (OPNING,X);         // OPPONENT'S
                BNE     (END);              // MOVE OK?
                DEX;
                LDAf    (OPNING,X);         // GET NEXT
                STA     (DIS1);             // CANNED
                DEX;                        // OPENING MOVE
                LDAf    (OPNING,X);
                STA     (DIS3);             // DISPLAY IT
                DEX;
                STX     (OMOVE);            // MOVE IT
                BNE     (MV2);              // (JMP)
//
END_            LDAi    (0xFF);             // *ADD - STOP CANNED MOVES
                STA     (OMOVE);            // FLAG OPENING
NOOPEN_         LDXi    (0x0C);             // FINISHED
                STX     (STATE);            // STATE=C
                STX     (BESTV);            // CLEAR BESTV
                LDXi    (0x14);             // GENERATE P
                JSR     (GNMX);             // MOVES
//
                LDXi    (0x04);             // STATE=4
                STX     (STATE);            // GENERATE AND
                JSR     (GNMZ);             // TEST AVAILABLE
//                                             MOVES
//
                LDX     (BESTV);            // GET BEST MOVE
                CPXi    (0x0F);             // IF NONE
                BCC     (MATE);             // OH OH!
//
MV2_            LDX     (BESTP);            // MOVE
                LDAx    (BOARD,X);          // THE
                STA     (BESTV);            // BEST
                STX     (PIECE);            // MOVE
                LDA     (BESTM);
                STA     (SQUARE);           // AND DISPLAY
                JSR     (MOVE);             // IT
                JSR_    ("__dosaveposition__"); // save for undo
                JSR_    ("__showboard__");  // display the whole board
                JMP     (CHESS_BEGIN);
//
MATE_           LDAi    (0xFF);             // RESIGN
                RTS;                        // OR STALEMATE
//
//       SUBROUTINE TO ENTER THE
//       PLAYER'S MOVE
```

```
//
DISMV_          LDXi    (0x04);             // ROTATE
DROL_           ASL     (DIS3);             // KEY
                ROL     (DIS2);             // INTO
                DEX;                        // DISPLAY
                BNE     (DROL);             //
                ORA_    (DIS3);
                STA     (DIS3);
                STA     (SQUARE);
                RTS_;
//
//       THE FOLLOWING SUBROUTINE ASSIGNS
//       A VALUE TO THE MOVE UNDER
//       CONSIDERATION AND RETURNS IT IN
//       THE ACCUMULATOR
//
STRATGY_        CLC;
                LDAi_   (0x80);
                ADC     (WMOB);             // PARAMETERS
                ADC     (WMAXC);            // WITH WEIGHT
                ADC     (WCC);              // OF 0.25
                ADC     (WCAP1);
                ADC     (WCAP2);
                SEC;
                SBC     (PMAXC);
                SBC     (PCC);
                SBC     (BCAP0);
                SBC     (BCAP1);
                SBC     (BCAP2);
                SBC     (PMOB);
                SBC     (BMOB);
                BCS     (POS);              // UNDERFLOW
                LDAi_   (0x00);             // PREVENTION
POS_            LSR;
                CLC;                        // **************
                ADCi    (0x40);
                ADC     (WMAXC);            // PARAMETERS
                ADC     (WCC);              // WITH WEIGHT
                SEC;                        // OF 0.5
                SBC     (BMAXC);
                LSR;                        // **************
                CLC;
                ADCi    (0x90);
                ADC     (WCAP0);            // PARAMETERS
                ADC     (WCAP0);            // WITH WEIGHT
                ADC     (WCAP0);            // OF 1.0
                ADC     (WCAP0);
                ADC     (WCAP1);
                SEC;                        // [UNDER OR OVER-
                SBC     (BMAXC);            // FLOW MAY OCCUR
                SBC     (BMAXC);            // FROM THIS
                SBC     (BMCC);             // SECTION]
                SBC     (BMCC);
                SBC     (BCAP1);
                LDX     (SQUARE);           // ***************
                CPXi    (0x33);
                BEQ     (POSN);             // POSITION
                CPXi    (0x34);             // BONUS FOR
                BEQ     (POSN);             // MOVE TO
```

```
                CPXi    (0x22);             // CENTRE
                BEQ     (POSN);             // OR
                CPXi    (0x25);             // OUT OF
                BEQ     (POSN);             // BACK RANK
                LDX     (PIECE);
                BEQ     (NOPOSN);
                LDYx    (BOARD,X);
                CPYi    (0x10);
                BPL     (NOPOSN);
POSN_           CLC;
                ADCi    (0x02);
NOPOSN_         JMP     (CKMATE);           // CONTINUE
//
//
POUT_           LDA_   (DIS1);
                fprintf(f,"      gosub __hexbyte__\r\n");
                fprintf(f,"      sertxd (\" \")\r\n");
                LDA_   (DIS2);
                fprintf(f,"      gosub __hexbyte__\r\n");
                fprintf(f,"      sertxd (\" \")\r\n");
                LDA_   (DIS3);
                fprintf(f,"      gosub __hexbyte__\r\n");
                fprintf(f,"      sertxd (cr,lf)\r\n");
                RTS_;

KIN_            LDAi    ('?');
                fprintf(f,"      sertxd (reg_a)\r\n");
                fprintf(f,"      serrxd reg_a\r\n");
                RTS_;
//
//
RESTART_CHESS_  fprintf(f,"      reset\r\n");
}

int main(int argc, char *argv[])
{
   init();
   chess();
   done();
   return 0;
}
```

Appendix E – Generated PICAXE Source Code for 20X2

```
#rem
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************

  Kim-1 MicroChess (c) 1976-2005 Peter Jennings, www.benlo.com
  6502 emulation   (c) 2005 Bill Forster
  20X2 emulation   (c) 2015 Ian Mitchell

  Runs an emulation of the Kim-1 Microchess on the PICAXE 20X2
  microcontroller. Based on an idea from Bill Forster to emulate
  6502 microprocessor instructions in C. The program is created
  by running 20X2Microchess.exe. This file (20X2Microchess.bas)
  is generated and can be uploaded to a 20X2.

**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************

 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:
 1. Redistributions of source code must retain the above copyright
       notice, this list of conditions and the following disclaimer.
 2. Redistributions in binary form must reproduce the above copyright
       notice, this list of conditions and the following disclaimer in the
       documentation and/or other materials provided with the distribution.
 3. The name of the author may not be used to endorse or promote products
       derived from this software without specific prior written permission.

 THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS OR
 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
 IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES LOSS OF USE,
 DATA, OR PROFITS OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#endrem

#picaxe 20x2
setfreq m64
symbol reg_a = b0
symbol reg_x = b1
symbol reg_y = b2
symbol reg_v = b3
symbol reg_f = b4
symbol reg_cy = b5
symbol reg_fc = w2
symbol temp = b6
```

```
symbol _row = b7
symbol _col = b8
symbol _loc = b9
symbol _pindex = b10
symbol _reverse = b11
symbol _p = b12

      gosub __read_static_data__
      reg_a = 0x00
      put 0x20,reg_a
      reg_a = 0xff
      put 0x21,reg_a
      reg_a = 0x00
      put 0x29,reg_a
      reg_x = 0x1f
      reg_a = 0xcc
_INITCLEAR_:
      put reg_x,reg_a
      dec reg_x : reg_f = reg_x
      if reg_f<0x80 then goto _INITCLEAR_
_CHESS_BEGIN_:
      reg_x = 0x7f
      ptr = reg_x
      reg_x = 0x60
      put 0x24,reg_x
      gosub _POUT_
      gosub _KIN_
      reg_a = reg_a&0x5f
      reg_fc = reg_a+0xa8 ; CMPi(0x58)
      if reg_f!=0 then goto _TESTLEVEL2_
      reg_a = 0x00
      put 0x20,reg_a
      reg_a = 0xff
      put 0x21,reg_a
      reg_a = 0x11
      goto _CLDSP2_
_TESTLEVEL2_:
      reg_fc = reg_a+0xa7 ; CMPi(0x59)
      if reg_f!=0 then goto _TESTLEVEL3_
      reg_a = 0x00
      put 0x20,reg_a
      reg_a = 0xfb
      put 0x21,reg_a
      reg_a = 0x22
      goto _CLDSP2_
_TESTLEVEL3_:
      reg_fc = reg_a+0xa6 ; CMPi(0x5a)
      if reg_f!=0 then goto _TESTSAVE_
      reg_a = 0x08
      put 0x20,reg_a
      reg_a = 0xfb
      put 0x21,reg_a
      reg_a = 0x33
      goto _CLDSP2_
_TESTSAVE_:
      reg_fc = reg_a+0xad ; CMPi(0x53)
      if reg_f!=0 then goto _TESTLOAD_
      gosub __saveposition__
```

```
      reg_a = 0x55
      goto _CLDSP2_
_TESTLOAD_:
      reg_fc = reg_a+0xb4 ; CMPi(0x4c)
      if reg_f!=0 then goto _TESTRESTORE_
      gosub __loadposition__
      reg_a = 0x88
      goto _CLDSP2_
_TESTRESTORE_:
      reg_fc = reg_a+0xae ; CMPi(0x52)
      if reg_f!=0 then goto _TESTUNDO_
      gosub __restoreposition__
      reg_a = 0x88
      goto _CLDSP2_
_TESTUNDO_:
      reg_fc = reg_a+0xab ; CMPi(0x55)
      if reg_f!=0 then goto _SETUP_
      gosub __undoposition__
      reg_a = 0x88
      goto _CLDSP2_
_SETUP_:
      reg_a = reg_a&0x4f
      reg_fc = reg_a+0xbd ; CMPi(0x43)
      if reg_f!=0 then goto _NOSET_
      reg_x = 0x1f
_WHSET_:
      reg_f = reg_x+0x10 : peek reg_f,reg_a : reg_f = reg_a
      put reg_x,reg_a
      dec reg_x : reg_f = reg_x
      if reg_f<0x80 then goto _WHSET_
      reg_x = 0x1b
      put 0x2a,reg_x
      reg_a = 0x00
      put 0x29,reg_a
      gosub __dosaveposition__
      reg_a = 0xcc
      goto _CLDSP_
_NOSET_:
      reg_fc = reg_a+0xbb ; CMPi(0x45)
      if reg_f!=0 then goto _NOREV_
      gosub _REVERSE_
      reg_cy = 1
      reg_a = 0x01
      get 0x29,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      put 0x29,reg_a
      gosub __dosaveposition__
      reg_a = 0xee
      goto _CLDSP_
_NOREV_:
      reg_fc = reg_a+0xc0 ; CMPi(0x40)
      if reg_f!=0 then goto _NOGO_
      @ptrdec = 0 : goto _GO_
_00:  gosub __dosaveposition__
_CLDSP_:
      gosub __showboard__
_CLDSP2_:
      put 0x43,reg_a
      put 0x42,reg_a
```

```
      put 0x41,reg_a
      goto _CHESS_BEGIN_
_NOGO_:
      reg_fc = reg_a+0xf3 ; CMPi(0x0d)
      if reg_f!=0 then goto _NOMV_
      @ptrdec = 1 : goto _MOVE_
_01:  gosub __showboard__
      goto _DISP_
_NOMV_:
      reg_fc = reg_a+0xbf ; CMPi(0x41)
      if reg_f=0 then goto _DONE_
      goto _INPUT_
_DONE_:
      goto _RESTART_CHESS_
_JANUS_:
      get 0x27,reg_x : reg_f = reg_x
      if reg_f>=0x80 then goto _NOCOUNT_
      get 0x22,reg_a : reg_f = reg_a
      if reg_f=0 then goto _OVER_
      reg_fc = reg_x+0xf8 ; CPXi(0x08)
      if reg_f!=0 then goto _OVER_
      get 0x34,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_f=0 then goto _XRT_
_OVER_:
      temp = reg_x+0x31 : get temp,reg_f : inc reg_f : put temp,reg_f
      reg_fc = reg_a+0xff ; CMPi(0x01)
      if reg_f!=0 then goto _NOQ_
      temp = reg_x+0x31 : get temp,reg_f : inc reg_f : put temp,reg_f
_NOQ_:
      if reg_v=0 then goto _NOCAP_
      reg_y = 0x0f : reg_f = reg_y
      get 0x23,reg_a : reg_f = reg_a
_ELOOP_:
      reg_f = reg_y+0x10 : get reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1
      if reg_f=0 then goto _FOUN_
      dec reg_y : reg_f = reg_y
      if reg_f<0x80 then goto _ELOOP_
_FOUN_:
      reg_f = reg_y+0x41 : peek reg_f,reg_a : reg_f = reg_a
      reg_f = reg_x+0x32 : get reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1
      if reg_cy=0 then goto _LESS_
      temp = reg_x+0x34 : put temp,reg_y
      temp = reg_x+0x32 : put temp,reg_a
_LESS_:
      reg_cy = 0
      temp = reg_v<<1|reg_cy : temp = reg_f>>7<<3|temp : temp = reg_f max
1<<2|temp : @ptrdec = temp
      reg_f = reg_x+0x33 : get reg_f,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      temp = reg_x+0x33 : put temp,reg_a
      inc ptr : temp = @ptr : reg_f = temp<<4&0xc0: reg_cy = temp&0x01 : reg_v =
temp>>1&0x01
_NOCAP_:
      reg_fc = reg_x+0xfc ; CPXi(0x04)
      if reg_f=0 then goto _ON4_
      if reg_f>=0x80 then goto _TREE_
_XRT_:
```

```
      goto __return__
_ON4_:
      get 0x36,reg_a : reg_f = reg_a
      put 0x2b,reg_a
      reg_a = 0x00 : reg_f = reg_a
      put 0x27,reg_a
      @ptrdec = 2 : goto _MOVE_
_02:  gosub _REVERSE_
      @ptrdec = 3 : goto _GNMZ_
_03:  gosub _REVERSE_
      reg_a = 0x08 : reg_f = reg_a
      put 0x27,reg_a
      @ptrdec = 4 : goto _GNM_
_04:  @ptrdec = 5 : goto _UMOVE_
_05:  goto _STRATGY_
_NOCOUNT_:
      reg_fc = reg_x+0x07 ; CPXi(0xf9)
      if reg_f!=0 then goto _TREE_
      get 0x10,reg_a : reg_f = reg_a
      get 0x23,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_f!=0 then goto _RETJ_
      reg_a = 0x00 : reg_f = reg_a
      put 0x26,reg_a
_RETJ_:
      goto __return__
_TREE_:
      if reg_v=0 then goto _RETJ_
      reg_y = 0x07 : reg_f = reg_y
      get 0x23,reg_a : reg_f = reg_a
_LOOPX_:
      reg_f = reg_y+0x10 : get reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1
      if reg_f=0 then goto _FOUNX_
      dec reg_y : reg_f = reg_y
      if reg_f=0 then goto _RETJ_
      if reg_f<0x80 then goto _LOOPX_
_FOUNX_:
      reg_f = reg_y+0x41 : peek reg_f,reg_a : reg_f = reg_a
      reg_f = reg_x+0x30 : get reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1
      if reg_cy=0 then goto _NOMAX_
      temp = reg_x+0x30 : put temp,reg_a
_NOMAX_:
      get 0x27,reg_f : dec reg_f : put 0x27,reg_f
      get 0x21,reg_a : reg_f = reg_a
      get 0x27,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_f=0 then goto _UPTREE_
      @ptrdec = 6 : goto _GENRM_
_UPTREE_:
_06:  get 0x27,reg_f : inc reg_f : put 0x27,reg_f
      goto __return__
_INPUT_:
      reg_fc = reg_a+0xf8 ; CMPi(0x08)
      if reg_cy!=0 then goto _ERROR_
      gosub _DISMV_
      goto _DISP_
_ERROR_:
      goto _CHESS_BEGIN_
_DISP_:
```

```
      reg_x = 0x1f : reg_f = reg_x
_SEARCH_:
      get reg_x,reg_a : reg_f = reg_a
      get 0x42,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_f=0 then goto _HERE_
      dec reg_x : reg_f = reg_x
      if reg_f<0x80 then goto _SEARCH_
_HERE_:
      put 0x43,reg_x
      put 0x22,reg_x
      goto _CHESS_BEGIN_
_GNMZ_:
      reg_x = 0x10 : reg_f = reg_x
_GNMX_:
      reg_a = 0x00 : reg_f = reg_a
_CLEAR_:
      temp = reg_x+0x2c : put temp,reg_a
      dec reg_x : reg_f = reg_x
      if reg_f<0x80 then goto _CLEAR_
_GNM_:
      reg_a = 0x10 : reg_f = reg_a
      put 0x22,reg_a
_NEWP_:
      get 0x22,reg_f : dec reg_f : put 0x22,reg_f
      if reg_f<0x80 then goto _NEX_
      goto __return__
_NEX_:
      gosub _RESET_
      get 0x22,reg_y : reg_f = reg_y
      reg_x = 0x08 : reg_f = reg_x
      put 0x28,reg_x
      reg_fc = reg_y+0xf8 ; CPYi(0x08)
      if reg_f<0x80 then goto _PAWN_
      reg_fc = reg_y+0xfa ; CPYi(0x06)
      if reg_f<0x80 then goto _KNIGHT_
      reg_fc = reg_y+0xfc ; CPYi(0x04)
      if reg_f<0x80 then goto _BISHOP_
      reg_fc = reg_y+0xff ; CPYi(0x01)
      if reg_f=0 then goto _QUEEN_
      if reg_f<0x80 then goto _ROOK_
_KING_:
      @ptrdec = 7 : goto _SNGMV_
_07:  if reg_f!=0 then goto _KING_
      if reg_f=0 then goto _NEWP_
_QUEEN_:
      @ptrdec = 8 : goto _LINE_
_08:  if reg_f!=0 then goto _QUEEN_
      if reg_f=0 then goto _NEWP_
_ROOK_:
      reg_x = 0x04 : reg_f = reg_x
      put 0x28,reg_x
_AGNR_:
      @ptrdec = 9 : goto _LINE_
_09:  if reg_f!=0 then goto _AGNR_
      if reg_f=0 then goto _NEWP_
_BISHOP_:
      @ptrdec = 10 : goto _LINE_
_10:  get 0x28,reg_a : reg_f = reg_a
      reg_fc = reg_a+0xfc ; CMPi(0x04)
```

```
      if reg_f!=0 then goto _BISHOP_
      if reg_f=0 then goto _NEWP_
_KNIGHT_:
      reg_x = 0x10 : reg_f = reg_x
      put 0x28,reg_x
_AGNN_:
      @ptrdec = 11 : goto _SNGMV_
_11:  get 0x28,reg_a : reg_f = reg_a
      reg_fc = reg_a+0xf8 ; CMPi(0x08)
      if reg_f!=0 then goto _AGNN_
      if reg_f=0 then goto _NEWP_
_PAWN_:
      reg_x = 0x06 : reg_f = reg_x
      put 0x28,reg_x
_P1_:
      @ptrdec = 12 : goto _CMOVE_
_12:  if reg_v=0 then goto _P2_
      if reg_f>=0x80 then goto _P2_
      @ptrdec = 13 : goto _JANUS_
_P2_:
_13:  gosub _RESET_
      get 0x28,reg_f : dec reg_f : put 0x28,reg_f
      get 0x28,reg_a : reg_f = reg_a
      reg_fc = reg_a+0xfb ; CMPi(0x05)
      if reg_f=0 then goto _P1_
_P3_:
      @ptrdec = 14 : goto _CMOVE_
_14:  if reg_v!=0 then goto _NEWP_
      if reg_f>=0x80 then goto _NEWP_
      @ptrdec = 15 : goto _JANUS_
_15:  get 0x23,reg_a : reg_f = reg_a
      reg_a = reg_a&0xf0
      reg_fc = reg_a+0xe0 ; CMPi(0x20)
      if reg_f=0 then goto _P3_
      goto _NEWP_
_SNGMV_:
      @ptrdec = 16 : goto _CMOVE_
_16:  if reg_f>=0x80 then goto _ILL1_
      @ptrdec = 17 : goto _JANUS_
_ILL1_:
_17:  gosub _RESET_
      get 0x28,reg_f : dec reg_f : put 0x28,reg_f
      goto __return__
_LINE_:
      @ptrdec = 18 : goto _CMOVE_
_18:  if reg_cy=0 then goto _OVL_
      if reg_v=0 then goto _LINE_
_OVL_:
      if reg_f>=0x80 then goto _ILL_
      temp = reg_v<<1|reg_cy : temp = reg_f>>7<<3|temp : temp = reg_f max
1<<2|temp : @ptrdec = temp
      @ptrdec = 19 : goto _JANUS_
_19:  inc ptr : temp = @ptr : reg_f = temp<<4&0xc0: reg_cy = temp&0x01 : reg_v =
temp>>1&0x01
      if reg_v=0 then goto _LINE_
_ILL_:
      gosub _RESET_
      get 0x28,reg_f : dec reg_f : put 0x28,reg_f
      goto __return__
```

```
_REVERSE_:
      reg_x = 0x0f
_ETC_:
      reg_cy = 1
      reg_f = reg_x+0x10 : get reg_f,reg_y
      reg_a = 0x77
      get reg_x,reg_f : reg_f = not reg_f: reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      temp = reg_x+0x10 : put temp,reg_a
      put reg_x,reg_y
      reg_cy = 1
      reg_a = 0x77
      get reg_x,reg_f : reg_f = not reg_f: reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      put reg_x,reg_a
      dec reg_x : reg_f = reg_x
      if reg_f<0x80 then goto _ETC_
      return
_CMOVE_:
      get 0x23,reg_a : reg_f = reg_a
      get 0x28,reg_x : reg_f = reg_x
      reg_cy = 0
      reg_f = reg_x+0x30 : peek reg_f,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a
= reg_f
      put 0x23,reg_a
      reg_a = reg_a&0x88 : reg_f = reg_a
      if reg_f!=0 then goto _ILLEGAL_
      get 0x23,reg_a : reg_f = reg_a
      reg_x = 0x20 : reg_f = reg_x
_LOOP_:
      dec reg_x : reg_f = reg_x
      if reg_f>=0x80 then goto _NO_
      get reg_x,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_f!=0 then goto _LOOP_
      reg_fc = reg_x+0xf0 ; CPXi(0x10)
      if reg_f>=0x80 then goto _ILLEGAL_
      reg_v = 1
      reg_a = 0x80 : reg_f = reg_a
      goto _SPX_
_NO_:
      reg_v = 0
_SPX_:
      get 0x27,reg_a : reg_f = reg_a
      if reg_f>=0x80 then goto _RETL_
      get 0x20,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_f<0x80 then goto _RETL_
      @ptrdec = reg_a
      temp = reg_v<<1|reg_cy : temp = reg_f>>7<<3|temp : temp = reg_f max
1<<2|temp : @ptrdec = temp
      reg_a = 0xf9 : reg_f = reg_a
      put 0x27,reg_a
      put 0x26,reg_a
      @ptrdec = 20 : goto _MOVE_
_20:  gosub _REVERSE_
      @ptrdec = 21 : goto _GNM_
_21:  @ptrdec = 22 : goto _RUM_
_22:  inc ptr : temp = @ptr : reg_f = temp<<4&0xc0: reg_cy = temp&0x01 : reg_v =
temp>>1&0x01
      inc ptr : reg_a = @ptr
```

```
      put 0x27,reg_a
      get 0x26,reg_a : reg_f = reg_a
      if reg_f>=0x80 then goto _RETL_
      reg_cy = 1
      reg_a = 0xff : reg_f = reg_a
      goto __return__
_RETL_:
      reg_cy = 0
      reg_a = 0x00 : reg_f = reg_a
      goto __return__
_ILLEGAL_:
      reg_a = 0xff : reg_f = reg_a
      reg_cy = 0
      reg_v = 0
      goto __return__
_RESET_:
      get 0x22,reg_x : reg_f = reg_x
      get reg_x,reg_a : reg_f = reg_a
      put 0x23,reg_a
      return
_GENRM_:
      @ptrdec = 23 : goto _MOVE_
_23:  gosub _REVERSE_
      @ptrdec = 24 : goto _GNM_
_RUM_:
_24:  gosub _REVERSE_
_UMOVE_:
      reg_x = ptr
      put 0x25,reg_x
      get 0x24,reg_x : reg_f = reg_x
      ptr = reg_x
      inc ptr : reg_a = @ptr
      put 0x28,reg_a
      inc ptr : reg_a = @ptr
      put 0x22,reg_a
      reg_x = reg_a : reg_f = reg_a
      inc ptr : reg_a = @ptr
      put reg_x,reg_a
      inc ptr : reg_a = @ptr
      reg_x = reg_a : reg_f = reg_a
      inc ptr : reg_a = @ptr
      put 0x23,reg_a
      put reg_x,reg_a
      goto _STRV_
_MOVE_:
      reg_x = ptr
      put 0x25,reg_x
      get 0x24,reg_x : reg_f = reg_x
      ptr = reg_x
      get 0x23,reg_a : reg_f = reg_a
      @ptrdec = reg_a
      reg_y = reg_a : reg_f = reg_a
      reg_x = 0x1f : reg_f = reg_x
_CHECK_:
      get reg_x,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_f=0 then goto _TAKE_
      dec reg_x : reg_f = reg_x
      if reg_f<0x80 then goto _CHECK_
      reg_x = 0x44
```

```
_TAKE_:
      reg_a = 0xcc : reg_f = reg_a
      put reg_x,reg_a
      reg_a = reg_x : reg_f = reg_x
      @ptrdec = reg_a
      get 0x22,reg_x : reg_f = reg_x
      get reg_x,reg_a : reg_f = reg_a
      put reg_x,reg_y
      @ptrdec = reg_a
      reg_a = reg_x : reg_f = reg_x
      @ptrdec = reg_a
      get 0x28,reg_a : reg_f = reg_a
      @ptrdec = reg_a
_STRV_:
      reg_x = ptr
      put 0x24,reg_x
      get 0x25,reg_x : reg_f = reg_x
      ptr = reg_x
      goto __return__
_CKMATE_:
      get 0x32,reg_x : reg_f = reg_x
      reg_f = 0+0x41 : peek reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_x+reg_f+1
      if reg_f!=0 then goto _NOCHEK_
      reg_a = 0x00 : reg_f = reg_a
      goto _RETV_
_NOCHEK_:
      get 0x31,reg_x : reg_f = reg_x
      if reg_f!=0 then goto _RETV_
      get 0x3c,reg_x : reg_f = reg_x
      if reg_f!=0 then goto _RETV_
      reg_a = 0xff : reg_f = reg_a
_RETV_:
      reg_x = 0x04 : reg_f = reg_x
      put 0x27,reg_x
      get 0x42,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+1
      if reg_cy=0 then goto _RETP_
      if reg_f=0 then goto _RETP_
      put 0x42,reg_a
      get 0x22,reg_a
      put 0x43,reg_a
      get 0x23,reg_a
      put 0x41,reg_a
_RETP_:
      reg_a = 0x2e
      sertxd (reg_a)
      goto __return__
_GO_:
      get 0x2a,reg_x : reg_f = reg_x
      if reg_f>=0x80 then goto _NOOPEN_
      get 0x41,reg_a : reg_f = reg_a
      reg_f = reg_x+0x51 : peek reg_f,reg_f : reg_f = not reg_f : reg_fc =
reg_a+reg_f+1
      if reg_f!=0 then goto _END_
      dec reg_x : reg_f = reg_x
      reg_f = reg_x+0x51 : peek reg_f,reg_a : reg_f = reg_a
      put 0x43,reg_a
      dec reg_x : reg_f = reg_x
      reg_f = reg_x+0x51 : peek reg_f,reg_a : reg_f = reg_a
```

```
      put 0x41,reg_a
      dec reg_x : reg_f = reg_x
      put 0x2a,reg_x
      if reg_f!=0 then goto _MV2_
_END_:
      reg_a = 0xff : reg_f = reg_a
      put 0x2a,reg_a
_NOOPEN_:
      reg_x = 0x0c : reg_f = reg_x
      put 0x27,reg_x
      put 0x42,reg_x
      reg_x = 0x14 : reg_f = reg_x
      @ptrdec = 25 : goto _GNMX_
_25:  reg_x = 0x04 : reg_f = reg_x
      put 0x27,reg_x
      @ptrdec = 26 : goto _GNMZ_
_26:  get 0x42,reg_x : reg_f = reg_x
      reg_fc = reg_x+0xf1 ; CPXi(0x0f)
      if reg_cy=0 then goto _MATE_
_MV2_:
      get 0x43,reg_x : reg_f = reg_x
      get reg_x,reg_a : reg_f = reg_a
      put 0x42,reg_a
      put 0x22,reg_x
      get 0x41,reg_a : reg_f = reg_a
      put 0x23,reg_a
      @ptrdec = 27 : goto _MOVE_
_27:  gosub __dosaveposition__
      gosub __showboard__
      goto _CHESS_BEGIN_
_MATE_:
      reg_a = 0xff : reg_f = reg_a
      goto __return__
_DISMV_:
      reg_x = 0x04 : reg_f = reg_x
_DROL_:
      get 0x41,reg_f : reg_cy = reg_f>>7 : reg_f = reg_f<<1 : put 0x41,reg_f
      get 0x42,reg_f : temp = reg_f>>7 : reg_f = reg_f<<1|reg_cy : put 0x42,reg_f
: reg_cy = temp
      dec reg_x : reg_f = reg_x
      if reg_f!=0 then goto _DROL_
      get 0x41,reg_f : reg_a = reg_a|reg_f
      put 0x41,reg_a
      put 0x23,reg_a
      return
_STRATGY_:
      reg_cy = 0
      reg_a = 0x80
      get 0x39,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0x3a,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0x3b,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0x2f,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0x2d,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      reg_cy = 1
      get 0x3e,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0x3f,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
```

```
      get 0x30,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0x2e,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0x2c,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0x3d,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0x31,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      if reg_cy!=0 then goto _POS_
      reg_a = 0x00
_POS_:
      reg_cy = reg_a&0x01 : reg_a = reg_a>>1 : reg_f = reg_a
      reg_cy = 0
      reg_fc = reg_a+0x40+reg_cy : reg_a = reg_f
      get 0x3a,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0x3b,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      reg_cy = 1
      get 0x32,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      reg_cy = reg_a&0x01 : reg_a = reg_a>>1 : reg_f = reg_a
      reg_cy = 0
      reg_fc = reg_a+0x90+reg_cy : reg_a = reg_f
      get 0x2b,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0x2b,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0x2b,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0x2b,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      get 0x2f,reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a = reg_f
      reg_cy = 1
      get 0x32,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0x32,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0x33,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0x33,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0x2e,reg_f : reg_f = not reg_f : reg_fc = reg_a+reg_f+reg_cy : reg_a =
reg_f
      get 0x23,reg_x : reg_f = reg_x
      reg_fc = reg_x+0xcd ; CPXi(0x33)
      if reg_f=0 then goto _POSN_
      reg_fc = reg_x+0xcc ; CPXi(0x34)
      if reg_f=0 then goto _POSN_
      reg_fc = reg_x+0xde ; CPXi(0x22)
      if reg_f=0 then goto _POSN_
      reg_fc = reg_x+0xdb ; CPXi(0x25)
      if reg_f=0 then goto _POSN_
      get 0x22,reg_x : reg_f = reg_x
      if reg_f=0 then goto _NOPOSN_
      get reg_x,reg_y : reg_f = reg_y
      reg_fc = reg_y+0xf0 ; CPYi(0x10)
      if reg_f<0x80 then goto _NOPOSN_
_POSN_:
      reg_cy = 0
      reg_fc = reg_a+0x02+reg_cy : reg_a = reg_f
_NOPOSN_:
      goto _CKMATE_
```

```
_POUT_:
      get 0x43,reg_a
      gosub __hexbyte__
      sertxd (" ")
      get 0x42,reg_a
      gosub __hexbyte__
      sertxd (" ")
      get 0x41,reg_a
      gosub __hexbyte__
      sertxd (cr,lf)
      return
_KIN_:
      reg_a = 0x3f : reg_f = reg_a
      sertxd (reg_a)
      serrxd reg_a
      return
_RESTART_CHESS_:
      reset
__hexbyte__:
      temp = reg_a>>4+"0"
      gosub __nybble__
      temp = reg_a&0x0f+"0"
__nybble__:
      if temp>"9" then : temp = temp+7 : endif
      sertxd (temp)
      return
__showboard__:
      gosub __backupposition__
      sertxd (cr,lf)
      gosub __rownum__
      gosub __line__
      get 0x29,_reverse
      for _row=0 to 7
         sertxd (#_row,"0|")
         for _col = 0 to 7
            _loc = _row<<4+_col
            for _pindex = 0 to 0x1f
               get _pindex,_p
               if _p=_loc then
                  _p = _pindex>>4^_reverse
                  if _p=0 then sertxd ("W") : else : sertxd ("B") : endif
                  _p = _pindex&0x0f
                  lookup _p,("KQRRBBNNPPPPPPPP"),_p
                  sertxd (_p)
                  goto __next_location__
               endif
            next
            ; not found
            _p = _row^_col&1
            if _p=1 then : sertxd ("**") : else : sertxd ("  ") : endif
            __next_location__:
            sertxd ("|")
         next
         sertxd (#_row,"0",cr,lf)
         gosub __line__
      next
      gosub __rownum__
      return
__line__:
```

```
      sertxd ("  ") : for _p=1 to 25 : sertxd ("-") : next : sertxd (cr,lf)
      return
__rownum__:
      sertxd ("  ") : for _p=0 to 7 : sertxd (" 0",#_p) : next : sertxd (cr,lf)
      return
__return__:
      inc ptr
      branch
@ptr,(_00,_01,_02,_03,_04,_05,_06,_07,_08,_09,_10,_11,_12,_13,_14,_15,_16,_17,_18,
_19,_20,_21,_22,_23,_24,_25,_26,_27)
__saveposition__:
      write 0xff,0xff
      _loc = 0x64
__saveposition0__:
      ;_loc has eeprom address
      for temp = 0 to 0x1f
         get temp,_p
         _pindex =_loc+temp
         write _pindex,_p
      next
      get 0x29,_p
      inc _pindex
      write _pindex,_p
      get 0x20,_p
      inc _pindex
      write _pindex,_p
      get 0x21,_p
      inc _pindex
      write _pindex,_p
      return
__loadposition0__:
      ;_loc has eeprom address
      for temp = 0 to 0x1f
         _pindex =_loc+temp
         read _pindex,_p
         put temp,_p
      next
      inc _pindex
      read _pindex,_p
      put 0x29,_p
      inc _pindex
      read _pindex,_p
      put 0x20,_p
      inc _pindex
      read _pindex,_p
      put 0x21,_p
      return
__loadposition__:
      read 0xff,temp
      if temp!=0xff then
         sertxd (cr,lf,"Save first.",cr,lf)
         return
      endif
      _loc = 0x64
      gosub __loadposition0__
      goto __showboard__
__backupposition__:
      write 0xfe,0xff
      _loc = 0x8c
```

```
      goto __saveposition0__
__restoreposition__:
      read 0xfe,temp
      if temp!=0xff then
         sertxd (cr,lf,"Can't restore.",cr,lf)
         return
      endif
      _loc = 0x8c
      gosub __loadposition0__
      goto __showboard__
__dosaveposition__:
      write 0xfd,0xff
      _loc = 0xb4
      goto __saveposition0__
__undoposition__:
      read 0xfd,temp
      if temp!=0xff then
         sertxd (cr,lf,"Can't undo.",cr,lf)
         return
      endif
      _loc = 0xb4
      gosub __loadposition0__
      goto __showboard__
__read_static_data__:
      for temp = 0 to 0x5c
         read temp,_p
         _loc = temp+0x10
         poke _loc,_p
      next
      return

      eeprom (0x03) ;SETW: 0x10
      eeprom (0x04) ;SETW: 0x11
      eeprom (0x00) ;SETW: 0x12
      eeprom (0x07) ;SETW: 0x13
      eeprom (0x02) ;SETW: 0x14
      eeprom (0x05) ;SETW: 0x15
      eeprom (0x01) ;SETW: 0x16
      eeprom (0x06) ;SETW: 0x17
      eeprom (0x10) ;SETW: 0x18
      eeprom (0x17) ;SETW: 0x19
      eeprom (0x11) ;SETW: 0x1a
      eeprom (0x16) ;SETW: 0x1b
      eeprom (0x12) ;SETW: 0x1c
      eeprom (0x15) ;SETW: 0x1d
      eeprom (0x14) ;SETW: 0x1e
      eeprom (0x13) ;SETW: 0x1f
      eeprom (0x73) ;SETW: 0x20
      eeprom (0x74) ;SETW: 0x21
      eeprom (0x70) ;SETW: 0x22
      eeprom (0x77) ;SETW: 0x23
      eeprom (0x72) ;SETW: 0x24
      eeprom (0x75) ;SETW: 0x25
      eeprom (0x71) ;SETW: 0x26
      eeprom (0x76) ;SETW: 0x27
      eeprom (0x60) ;SETW: 0x28
      eeprom (0x67) ;SETW: 0x29
      eeprom (0x61) ;SETW: 0x2a
      eeprom (0x66) ;SETW: 0x2b
```

```
eeprom (0x62) ;SETW: 0x2c
eeprom (0x65) ;SETW: 0x2d
eeprom (0x64) ;SETW: 0x2e
eeprom (0x63) ;SETW: 0x2f
eeprom (0x00) ;MOVEX: 0x30
eeprom (0xf0) ;MOVEX: 0x31
eeprom (0xff) ;MOVEX: 0x32
eeprom (0x01) ;MOVEX: 0x33
eeprom (0x10) ;MOVEX: 0x34
eeprom (0x11) ;MOVEX: 0x35
eeprom (0x0f) ;MOVEX: 0x36
eeprom (0xef) ;MOVEX: 0x37
eeprom (0xf1) ;MOVEX: 0x38
eeprom (0xdf) ;MOVEX: 0x39
eeprom (0xe1) ;MOVEX: 0x3a
eeprom (0xee) ;MOVEX: 0x3b
eeprom (0xf2) ;MOVEX: 0x3c
eeprom (0x12) ;MOVEX: 0x3d
eeprom (0x0e) ;MOVEX: 0x3e
eeprom (0x1f) ;MOVEX: 0x3f
eeprom (0x21) ;MOVEX: 0x40
eeprom (0x0b) ;POINTS: 0x41
eeprom (0x0a) ;POINTS: 0x42
eeprom (0x06) ;POINTS: 0x43
eeprom (0x06) ;POINTS: 0x44
eeprom (0x04) ;POINTS: 0x45
eeprom (0x04) ;POINTS: 0x46
eeprom (0x04) ;POINTS: 0x47
eeprom (0x04) ;POINTS: 0x48
eeprom (0x02) ;POINTS: 0x49
eeprom (0x02) ;POINTS: 0x4a
eeprom (0x02) ;POINTS: 0x4b
eeprom (0x02) ;POINTS: 0x4c
eeprom (0x02) ;POINTS: 0x4d
eeprom (0x02) ;POINTS: 0x4e
eeprom (0x02) ;POINTS: 0x4f
eeprom (0x02) ;POINTS: 0x50
eeprom (0x99) ;OPNING: 0x51
eeprom (0x25) ;OPNING: 0x52
eeprom (0x0b) ;OPNING: 0x53
eeprom (0x25) ;OPNING: 0x54
eeprom (0x01) ;OPNING: 0x55
eeprom (0x00) ;OPNING: 0x56
eeprom (0x33) ;OPNING: 0x57
eeprom (0x25) ;OPNING: 0x58
eeprom (0x07) ;OPNING: 0x59
eeprom (0x36) ;OPNING: 0x5a
eeprom (0x34) ;OPNING: 0x5b
eeprom (0x0d) ;OPNING: 0x5c
eeprom (0x34) ;OPNING: 0x5d
eeprom (0x34) ;OPNING: 0x5e
eeprom (0x0e) ;OPNING: 0x5f
eeprom (0x52) ;OPNING: 0x60
eeprom (0x25) ;OPNING: 0x61
eeprom (0x0d) ;OPNING: 0x62
eeprom (0x45) ;OPNING: 0x63
eeprom (0x35) ;OPNING: 0x64
eeprom (0x04) ;OPNING: 0x65
eeprom (0x55) ;OPNING: 0x66
```

```
eeprom (0x22) ;OPNING: 0x67
eeprom (0x06) ;OPNING: 0x68
eeprom (0x43) ;OPNING: 0x69
eeprom (0x33) ;OPNING: 0x6a
eeprom (0x0f) ;OPNING: 0x6b
eeprom (0xcc) ;OPNING: 0x6c
```